Titre: Title:	Stratégies de mise en antémémoire dans les réseaux ad hoc
Auteur: Author:	Thérence Houngbadji
Date:	2005
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Houngbadji, T. (2005). Stratégies de mise en antémémoire dans les réseaux ad hoc [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7629/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7629/
Directeurs de recherche: Advisors:	
Programme: Program:	Unspecified

UNIVERSITÉ DE MONTRÉAL

STRATÉGIES DE MISE EN ANTÉMÉMOIRE DANS LES RÉSEAUX AD HOC

THÉRENCE HOUNGBADJI
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
Décembre 2005

Your file Votre référence
ISBN: 978-0-494-16797-7
Our file Notre référence
ISBN: 978-0-494-16797-7

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

STRATÉGIES DE MISE EN ANTÉMÉMOIRE DANS LES RÉSEAUX AD HOC

présenté par : HOUNGBADJI Thérence
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. BILODEAU Guillaume-Alexandre, Ph.D., président
M. PIERRE Samuel, Ph.D., membre et directeur de recherche
M. QUINTERO Alejandro, Doct., membre et codirecteur
M. BENSLIMANE Abderrahim, Doct., membre

REMERCIEMENTS

Je voudrais tout d'abord remercier le directeur de la Fondation de l'œuvre Saint Justin en Suisse, M. Nicolas Scherrer, pour tout son appui.

Je remercie aussi mon directeur de recherche, M. Samuel Pierre pour son soutien indéfectible et ses encouragements tout au long de ce travail.

Je remercie aussi M. Alejandro Quintero mon codirecteur et M. Haidar Safa, pour leurs critiques et leurs conseils.

Enfin les membres du LARIM pour leurs collaborations et leurs soutiens.

RÉSUMÉ

Dans les réseaux ad hoc communément appelés MANET (*Mobile Ad hoc Network*), bien que le routage soit d'une grande importance, il existe également un autre élément d'intérêt tout aussi important qui est la mise en antémémoire. Une antémémoire est une partie de la mémoire d'une unité mobile dédiée pour le stockage de données fréquemment accédées par cette dernière.

En effet, les unités mobiles n'ont pas souvent la possibilité d'accéder facilement aux données localisées sur un serveur distant, à cause de leurs ressources limitées (bande passante faible, énergie limitée, faible portée radio, puissance de calcul faible, mémoire restreinte). Une relocalisation des données stockées sur le serveur distant à proximité des unités mobiles permettrait d'améliorer les performances de ces réseaux. Deux étapes permettent de mettre en place un mécanisme de mise en antémémoire : la répartition à moindre coût des données du serveur et l'invalidation d'antémémoire.

L'objectif de ce mémoire consiste à concevoir et développer une stratégie pour la mise en antémémoire de données tout en optimisant l'utilisation des ressources limitées des unités mobiles. De façon spécifique, la stratégie que nous proposons est une combinaison des techniques de mise de données en antémémoire et de réplication de données. Elle inclut une réplication partielle de données du serveur sur des unités mobiles clés, choisies suivant un profil et une fonction de coût. Cette dernière est une fonction dont l'objectif est de réduire le coût d'une unité mobile choisie pour la réplication, le délai d'accès aux données ainsi que leur stockage. La stratégie intègre aussi les méthodes d'accès et de compression de données.

Pour évaluer la performance de la solution proposée, nous l'avons implémentée et simulée à travers une série d'expériences, en mettant l'accent sur le délai moyen d'accès aux données ainsi que le débit efficace. Enfin, nous l'avons comparé avec d'autres stratégies telles que celle sans aucune mise en antémémoire, celle avec une mise en antémémoire systématique et une stratégie proposée dans la littérature, dénommée *Greedy-S*.

Les résultats obtenus montrent que le modèle de réplication partielle de données proposé offre une meilleure performance en termes de délai moyen d'accès aux données et du débit efficace de données, comparativement aux autres stratégies.

ABSTRACT

In commonly called ad hoc networks, although the routing is of great importance, there is also another element of interest such as data caching. A cache memory is part of the memory of a mobile unit dedicated for the storage of data frequently accessed. Indeed, the mobile units often do not have the possibility of easily reaching the data located on a remote server, because of their limited resources (weak bandwidth, limited energy, weak radio range, low computing power, restricted memory).

A relocalization of data stored on the remote server, near the mobile units, would make it possible to improve the performances of these networks. Two stages are required to install a mechanism of data caching: the distribution with lower cost of the server data, and the update or the invalidation of cache memory.

The objective of this thesis consists of conceiving and developing a strategy for the database caching, while optimizing the use of the limited resources of the mobile units. More specifically, the strategy that we propose is a combination of data caching and replication techniques.

Our approach includes a partial replication of server data on selected mobile units according to a profile and an optimization function. The latter is a function which minimizes in particular, the cost of selected mobile unit for the replication, the data access delay and their storage. The strategy integrates also the access methods and data compression.

To evaluate the performance of the proposed solution, we implemented it and simulated through a series of experiments, by having the focus on the data average access delay, the hit ratio and the throughput. Finally, we compared it with other strategies such as those without any caching mechanism, with a systematic caching, and a strategy suggested in the literature, named *Greedy-S*.

Our results show that the partial data replication model outperforms the other strategies, in terms of data average access delay and data throughput.

TABLES DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

CHAPITRE I

INTRODUCTION

Les réseaux ad hoc communément appelés MANET (*Mobile Ad hoc Network*), instances de réseaux sans fil, suscitent une attention particulière de la part de la communauté scientifique et de l'industrie des télécommunications, à cause de leur simplicité et rapidité de déploiement. Ils sont particulièrement adaptés pour les situations où l'installation d'infrastructures à grande échelle n'est pas possible, telles que les champs de bataille ainsi que les catastrophes géographiques. Les premiers champs de recherche sur ces types de réseaux se sont focalisés sur le développement de protocoles de routage dynamique. Bien que le routage soit d'une grande importance dans les réseaux ad hoc, il existe également un autre élément d'intérêt tout aussi important qu'est l'accès aux données des entités composant ce réseau. En effet, récemment ont été installés des centres d'informations pour permettre aux touristes en visite dans un musée de bénéficier d'un guide en ligne à l'aide de leurs terminaux personnels, ou aux voyageurs dans une gare d'accéder aux horaires de trains ou d'avions. Ces derniers sont ainsi des usagers mobiles qui, à un certain moment au cours de leur déplacement, peuvent ne plus se trouver à la portée du central d'information qui les dessert. Ils perdent ainsi l'accès à l'information requise en ce moment. Par contre, si le terminal de l'usager est équipé d'une antémémoire contenant l'information requise auparavant utilisée, il disposerait toujours de cette dernière sans avoir à en formuler la requête au central. C'est dans ce contexte qu'on fonde beaucoup d'espoir sur la mise de données en antémémoires pour améliorer l'accessibilité des terminaux mobiles aux services de données, ce qui constitue l'objet de ce mémoire. Dans ce chapitre d'introduction, nous allons présenter tout d'abord les concepts de base sur lesquels reposent les éléments de problématique reliés au sujet, puis nous formulerons nos objectifs de recherche. Finalement nous présenterons un plan de mémoire.

1.1 Définitions et concepts de base

Une antémémoire communément appelée *cache*, en anglais, est à l'origine une petite portion de la mémoire RAM d'un ordinateur utilisée pour faciliter l'accès aux données entre le processeur et les différents périphériques, en entretenant les données fréquemment accédées par ces derniers dans cette partie spécialisée de la mémoire. C'est une terminologie souvent appliquée au système processeur-mémoire, mais qui s'est étendue avec l'essor de l'accessibilité des données à travers un réseau. Dans les réseaux, c'est un dispositif matériel éventuellement associé à un composant logiciel dont l'objectif est de stocker localement des données afin de diminuer le délai de mise à disposition de celles-ci. Les antémémoires sont largement utilisées aujourd'hui notamment par Internet et donnent lieu aux techniques appelées *web caching* dont l'objectif est de stocker temporairement les documents pour un accès futur de l'usager [1].

La mise en antémémoire ou *caching* est une technique qui exploite la tendance des utilisateurs au sein d'une communauté donnée, d'accéder à un même document. Dans un tel environnement de mémorisation, une requête effectuée pour obtenir une information est directement transmise à un système d'antémémoires. Si l'information requise ne s'y trouve pas, on parle de *cache miss* et la requête est transférée vers l'hôte l'hébergeant. L'usager sera donc servi dépendamment de la bande passante dont dispose l'hôte pour lui transmettre la ressource demandée. La mise en antémémoire peut aussi s'effectuer dynamiquement, de telle sorte que les informations requises sont stockées au fur et à mesure en incluant également d'autres ressources qui n'ont fait l'objet d'aucune requête par l'usager ; ce dernier peut être servi très rapidement et complètement lors d'une prochaine requête sur la même ressource, auquel cas on parle de *cache hit*.

Plusieurs systèmes d'antémémoires peuvent également être configurés pour coopérer. Dans de tels systèmes, une antémémoire peut rediriger une demande pour une ressource demandée par l'usager vers une autre antémémoire plutôt que vers le serveur hôte de la ressource. Cette méthode est particulièrement efficace lorsque la bande

passante entre certains systèmes d'antémémoires est supérieure ou moins encombrée que celle qui est disponible entre l'antémémoire locale et le serveur hôte de l'information requise. Dans un tel environnement où plusieurs systèmes d'antémémoires coopèrent pour servir la requête de l'usager, on parle de *cooperative caching* [2] [3]. Une telle coopération résulte de la mise en place d'une stratégie de localisation des ressources fréquemment accédées par l'utilisateur, donnant ainsi lieu au concept de réplication de données.

La réplication de données est une technique permettant de garder une copie de la même information sur plusieurs serveurs de données ou dans plusieurs antémémoires, de manière à garantir un accès éventuellement hiérarchique lors d'une demande de la ressource. Elle est utile pour servir une communauté d'usagers, mais souffre d'un gaspillage de l'espace en antémémoire du fait que la copie de la même ressource peut se trouver dans toutes les antémémoires.

La mise en place d'antémémoires pour réduire l'usage abusif des ressources d'un réseau et pour améliorer les délais fait également intervenir un concept tout aussi important qu'est la consistance des données en antémémoire. En effet, à un certain moment, les données maintenues en antémémoires peuvent connaître des changements sur l'hôte les hébergeant, et donc devenir invalides. La consistance des données réfère à la possibilité pour la source de la donnée de mettre à jour à intervalle de temps régulier ou non les données maintenues en antémémoires par les usagers ou par les systèmes d'antémémoires. Cette mise à jour s'effectue sous deux formes :

- propagation : un objet qui connaît un changement dans la base de données est propagé vers toutes les antémémoires ;
- invalidation : quand l'objet connaît un changement, un message d'invalidation est envoyé à toutes les antémémoires, qui le marquent comme invalide. Une requête future sur cet objet ne pourra être servie que par le serveur de la donnée et non plus par l'antémémoire de l'usager.

Ces deux formes de mise à jour des données profitent souvent de la bande passante à disposition de la source de la donnée. Par ailleurs, la façon dont les données sont

stockées en antémémoires ou transmises fait également intervenir le concept de granularité.

La granularité réfère au niveau de détails des informations stockées en antémémoire ou transmises pendant la mise à jour. Elles peuvent être sous une forme compressée ou non, dépendamment de l'espace de stockage à disposition. Dans les réseaux filaires, son importance est souvent minimisée à cause de l'espace de stockage important dont disposent les infrastructures filaires, ainsi que leur bande passante relativement large.

Toute cette terminologie à l'origine utilisée pour les réseaux filaires, doit être réadaptée aux réseaux ad hoc dont l'architecture complètement distribuée est basée sur des entités mobiles servant de routeurs les unes aux autres et requérant des accès aux informations indépendamment de leur emplacement. L'accès aux informations hébergées par une source de données dans ces types de réseaux fait l'objet d'un intérêt de recherche de plus en plus grandissant, particulièrement en ce qui concerne les stratégies de mise en antémémoire, ainsi que les mécanismes de mise à jour des antémémoires.

1.2 Éléments de la problématique

Dans les réseaux sans fil, les utilisateurs équipés d'ordinateurs portables sans fil sont appelés à effectuer des requêtes sur une base de données à travers le médium sans fil pour obtenir des informations bien déterminées. En particulier, les réseaux ad hoc composés d'unités mobiles sont des systèmes à architectures distribuées dont le dynamisme de la topologie ainsi que les caractéristiques en font une aire fertile de recherche, en ce qui concerne l'accès aux données entreposées sur un serveur distant. Ainsi, pour être à même de garantir le succès aux services requis par l'usager, l'accessibilité aux données doit viser le plus grand nombre possible d'entre eux. Cependant, plusieurs défis liés aux fonctionnements des réseaux ad hoc nécessitent d'être pris en compte pour assurer cette accessibilité aux données. Au prime abord, les

problèmes intrinsèques aux réseaux ad hoc sont multiples, au nombre desquels on peut citer:

- une asymétrie de la communication, c'est-à-dire que la bande passante dans le sens descendant (serveur - client) est plus grande que celle du sens ascendant, de sorte que le client peut ne pas avoir la capacité de dialoguer avec le serveur ;
- les déconnexions fréquentes des usagers, c'est-à-dire que les terminaux dont disposent ceux-ci dans leur mobilité ne restent pas indéfiniment connectés, ceci dans le but d'économiser l'énergie déjà limitée dont dispose leur batterie ;
- la connexion, si elle est établie entre unités mobiles, n'est pas toujours fiable offrant du coup une qualité de service imprévisible.

Chacune de ces caractéristiques a un impact direct sur la manière dont la gestion de l'accès aux données peut être effectuée. L'asymétrie de la communication peut, à première vue, paraître comme un moyen d'éviter les faiblesses en énergie des unités mobiles, car elle fait du modèle de dissémination de données (*broadcast*) une technique attractive, au lieu d'attendre que l'unité mobile effectue une requête pour obtenir un service. Cependant, elle relève d'un gaspillage de ressources et nécessite d'être utilisée efficacement.

La faible bande passante ainsi que le motif de déconnexion des unités mobiles ont une influence évidente sur la manière dont les transactions doivent s'effectuer entre entités du réseau ad hoc.

Lorsque les usagers se déplacent d'une zone de service de données à une autre, ils doivent être pris en charge par le nouveau serveur de données pour éviter un gaspillage de ressources et des attentes prolongées. Il y a donc une nécessité de définir une procédure de prise en charge de la relève (*handoff*) des usagers.

Avec toutes ces restrictions dont font l'objet les réseaux ad hoc, la mise en antémémoire par les unités mobiles de données résidant sur un serveur distant apparaît comme une technique efficace pour améliorer les performances de l'accessibilité aux données dans un environnement mobile. Plusieurs accès aux données peuvent, comme dans le cas des réseaux filaires être servis directement à l'usager depuis son

antémémoire. Cela réduit considérablement la latence d'accès aux informations requises par l'usager et diminue le risque d'usage abusif de la bande passante. Cependant, le schéma classique utilisé dans les réseaux filaires où les usagers sont toujours connectés, ne pourrait pas convenir à un environnement aussi dynamique que celui des réseaux ad hoc, à cause de leur déconnexion fréquente et de leur mobilité incontrôlable.

Typiquement, pour les unités mobiles effectuant des requêtes sur des données entreposées sur un serveur, la validation des données qu'elles mettent en antémémoire est une opération coûteuse en bande passante.

Par ailleurs, si toutes les unités mobiles doivent effectuer à chaque fois des requêtes en direction d'un seul serveur de données, cela risquerait d'augmenter la charge du serveur ainsi que la latence d'accès à l'information requise. D'où un problème d'évolutivité, c'est-à-dire que si, à un certain moment, le nombre d'unités mobiles dans une cellule donnée devient important, il n'y aurait plus aucune garantie de service ni de qualité de service.

De même, si toutes les unités mobiles doivent disposer d'une antémémoire et stocker les mêmes données dans cette dernière, cela relèverait d'un gaspillage d'espace mémoire, posant du coup le problème de localisation d'antémémoires dans les réseaux ad hoc. Par exemple, on peut imaginer qu'un groupe de travail traitant du même sujet accède pratiquement tout le temps aux mêmes informations et constitue ainsi un scénario typique de gaspillage d'espace puisque chaque membre du groupe dispose d'une antémémoire pour stocker ces mêmes informations.

Pour stocker les données, les unités mobiles dont la capacité de calcul est limitée peuvent ne pas adopter des mécanismes de compression de données et verraient ainsi le cycle de renouvellement de leur antémémoire raccourcir, effet dû à un stockage abusif de l'information. Même disposant d'une antémémoire, les unités mobiles peuvent ne pas être complètement à même de satisfaire une requête sur la base de l'information qu'elles détiennent.

Les recherches effectuées sur la mise de données en antémémoire se sont surtout focalisées sur les mécanismes d'invalidation d'antémémoires, perçus comme des

moyens d'améliorer l'accessibilité des usagers aux données. Pourtant, le problème de la stratégie à adopter pour mettre à la disposition des usagers les données avant de les invalider demeure entier.

1.3 Objectifs de la recherche

Ce mémoire a justement pour objectif de concevoir une stratégie de mise de données en antémémoire dans les réseaux ad hoc, tout en optimisant l'utilisation des ressources limitées des unités mobiles. De manière spécifique, ce mémoire vise à :

- proposer une architecture pour la mise de données en antémémoire en se basant sur celles existantes ;
- concevoir des mécanismes pour la compression et le stockage des données ainsi qu'une méthode pour l'identification, l'association et la translation des directives utilisées pour accéder aux données ;
- évaluer la performance de l'architecture et des mécanismes proposés en considérant des métriques telles que le délai moyen d'accès aux objets, le débit de réponse aux requêtes, ainsi que le pourcentage de succès en antémémoire ou *hit ratio*.

1.4 Plan du mémoire

Le chapitre II fait le point sur les stratégies existantes pour la mise de données en antémémoire. Un état de l'art des stratégies de mise en antémémoire et de réplication de données y sera effectué.

Une fois les concepts de base définis à travers cet état de l'art, le chapitre III se focalisera sur la définition des stratégies et algorithmes proposés pour la mise de données en antémémoires. Par la suite, nous ferons une modélisation analytique de la stratégie proposée.

Dans le chapitre IV, nous effectuerons une mise en œuvre des stratégies proposées. Ce chapitre se divise en deux parties. Nous ferons au prime abord, une implémentation des stratégies proposées dans un simulateur. Par la suite, nous

produirons les résultats de cette implémentation puis effectuerons une analyse de la performance de ces stratégies.

Pour conclure, le chapitre V produira une synthèse des travaux en mettant l'accent sur les principaux résultats obtenus, sur les limitations de la stratégie conçue, ainsi que sur les améliorations possibles qu'on pourrait y apporter. Enfin, nous définirons une esquisse des recherches futures relatives au sujet traité dans le présent mémoire.

CHAPITRE II

MISE EN ANTÉMÉMOIRE

DANS LES RÉSAUX AD HOC

La conception d'une stratégie de mise de données en antémémoire pour les réseaux mobiles ad hoc nécessite au préalable une étude de celles déjà existantes. Dans ces types de réseaux, de nombreux usagers munis d'ordinateurs portables (PocketPC, Palm Pilot, Pager) effectuent des requêtes sur des serveurs de données à travers le médium sans fil pour accéder à une information précise indépendamment du temps et de l'espace. Ces ordinateurs portables, appelés abusivement unités mobiles (UM) à cause de la mobilité des usagers, sont limités en ressources telles que la durée de vie de leur batterie et la capacité de leur mémoire [4]. De plus, elles doivent se déconnecter à intervalles de temps plus ou moins réguliers et prolongés pour réduire leur consommation en énergie. De même, du fait de leur caractère nomade, ces UM peuvent ne pas rester connectées au même serveur de données. Toutes ces contraintes, que ce soit du côté des UM que de celui du médium ont un impact direct sur la manière dont les données doivent être gérées dans un environnement mobile.

La motivation menant à la conception d'une stratégie de mise de données en antémémoire vient du fait qu'elle permet une meilleure accessibilité aux données en réduisant les interactions entre serveur de données et UM. Cela permet ainsi de réduire la latence d'accès ainsi qu'un usage rationnel de la bande passante du médium sans fil. Dans ce chapitre, nous ferons un bref survol des réseaux ad hoc. En outre, nous effectuerons une analyse des problèmes de gestion d'antémémoire dans les réseaux ad hoc, puis présenterons les stratégies existantes pour cette gestion.

2.1 Réseaux ad hoc

Les réseaux ad hoc sont une collection composite ou non d'entités mobiles reliées entre elles par les ondes électromagnétiques, formant un réseau sans fil sans l'aide d'aucune infrastructure tierce ou d'une administration centrale de commutation. Chaque entité mobile agit en tant que routeur avec éventuellement des contraintes liées à un type de qualité de service. Les réseaux ad hoc peuvent fonctionner de manière isolée ou être connectés au réseau filaire. La Figure 2.1 présente une architecture de ce type de réseau.

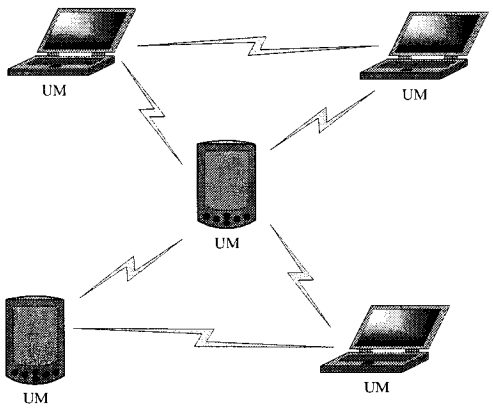


Figure 2.1 Architecture d'un réseau ad hoc

Les principales caractéristiques des réseaux ad hoc sont les suivantes:

- bande passante limitée à la disposition des unités mobiles ;
- topologie extrêmement dynamique ;
- batterie limitée ;
- taux d'erreurs élevés sur le médium de transmission ;
- rupture fréquente des chemins de transit de l'information.

Avec de telles restrictions, les UM doivent accéder à des données résidant sur un serveur de bases de données (ou un serveur WEB). En particulier, pour un réseau ad hoc de grande taille, l'accès aux informations peut devenir rapidement chaotique à cause de la concurrence d'accès des UM à une information vitale. De ce fait, un mécanisme de régulation de l'accès aux données tel une mise en antémémoire est nécessaire.

La mise en antémémoire de données est déjà largement adoptée dans le cas des réseaux filaires pour l'usage du protocole HTTP ainsi que dans les systèmes d'exploitation.

2.2 Gestion d'antémémoires dans les réseaux mobiles

Avec les avantages qu'offrent les schémas classiques de gestion d'antémémoires dans les réseaux filaires, les stratégies utilisées par ces derniers ne sont pas qualifiables pour un usage dans les réseaux sans fil. En effet, les UM effectuent des déconnexions fréquentes [5] pour économiser de l'énergie, si bien qu'il est difficile pour un serveur d'effectuer des mises à jour des données stockées par ces UM de façon permanente. D'autre part, des requêtes effectuées par des UM pour valider des données récemment mémorisées, peuvent obtenir des réponses lentes à cause de la limitation en bande passante. C'est dans ce contexte que, pour mieux résoudre les problèmes liés à l'accès aux données dans les réseaux mobiles, l'architecture d'accès aux données illustrée par la Figure 2.2 est proposée [6].

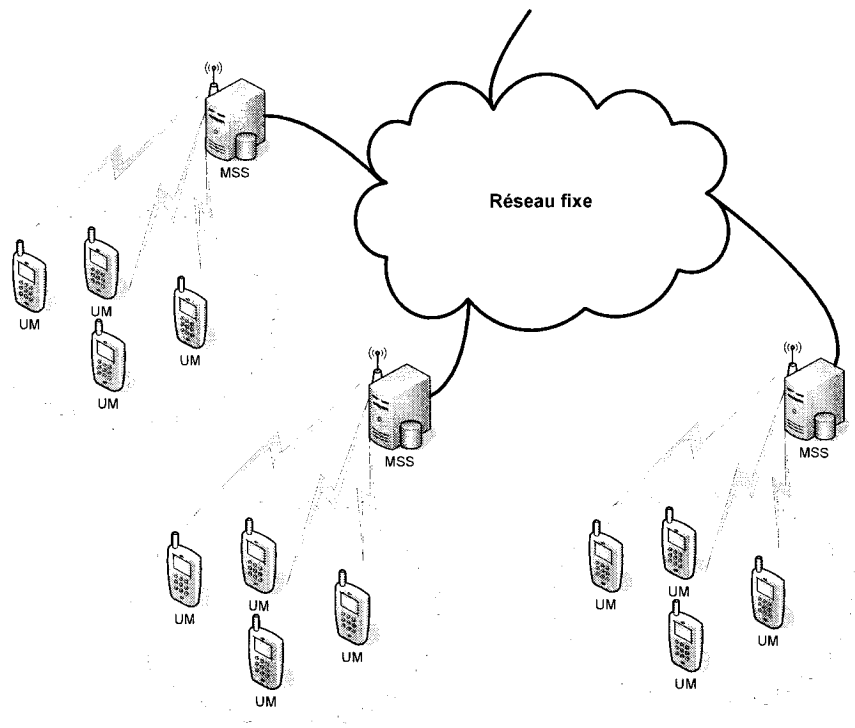


Figure 2.2 Architecture d'accès aux données d'un environnement mobile

Ce modèle largement repris dans la littérature [7-12] se compose d'entités fixes et mobiles. Les *MSS* (*Mobile Support Station*) sont des stations fixes, et sont supposées être équipées d'une interface sans fil qui leur permet de communiquer avec les UM se trouvant à leur portée radio, c'est-à-dire dans la cellule couverte par ces MSS. Les UM sont équipées d'un dispositif de stockage susceptible de garder de l'information même après une déconnexion (mémoire flash, disque dur). Elles accèdent à une base de données répliquée sur toutes les MSS. Cette base de données est régulièrement mise à jour de manière à garantir une consistance des données.

2.2.1 Problèmes de mise de données en antémémoires

Dans les systèmes de bases de données client-serveur traditionnels, il est relativement facile de maintenir une consistance des données détenues par le client dont la localisation est fixe et connue. De plus, la connexion du client au serveur de base de données change peu. À l'inverse, dans les réseaux mobiles, maintenir des antémémoires au sein de chaque UM n'est pas aussi simple comme dans le cas du modèle client-serveur des réseaux filaires, car les UM ont un motif de déconnexion aléatoire. Avec l'architecture d'accès aux données présentée à la Figure 2.2, il est impossible, avec les procédures conventionnelles des réseaux filaires pour une UM, de déterminer si les données disponibles dans son antémémoire permettent de répondre adéquatement à une requête. Elle pourrait ainsi être obligée de contacter le serveur de données pour récupérer de l'information additionnelle ou tout simplement d'attendre une éventuelle mise à jour de son antémémoire. Cela prolonge le délai d'accès à l'information, gaspille la bande passante de l'UM déjà très limitée ainsi que l'énergie de sa batterie.

L'objectif central de la mise de données en antémémoire sera donc d'essayer de minimiser le délai d'accès à l'information en limitant les interactions inutiles mais aussi de réduire le nombre de messages transmis pour les interactions nécessaires. Deux stratégies de mise de données en antémémoire au sein de l'UM sont décrites dans la littérature [13] : Stateful Server, et Stateless Server.

Stateful Server : le serveur de données connaît exactement toutes les UM se trouvant dans sa zone de couverture ainsi que l'état de leur antémémoire. Si un objet particulier mis en antémémoire par une UM connaît une mise à jour sur le serveur, ce dernier produit un message d'invalidation de cet objet qu'il envoie à cette UM. Cette stratégie est exactement la même que celle client-serveur classique. Les UM doivent donc prévenir le serveur de leur état dans le temps (déconnexions, reconnexions, handoff). Cette approche présente de sérieuses faiblesses car elle requiert beaucoup trop d'interactions entre UM et serveurs. D'autre part, si le nombre d'UM est élevé, l'accès aux données à travers le médium sans fil peut devenir immédiatement chaotique à cause du nombre d'interactions pouvant devenir rapidement important.

Stateless Server : dans cette approche, le serveur n'a aucune information sur les UM se trouvant à un moment donné dans sa zone de couverture et donc aucune information sur l'état de leur antémémoire. Le serveur peut servir des données à toutes les UM sous sa tutelle sans aucune contrainte, et ceci de manière synchrone ou non. Cette approche reflète plus la réalité car elle tient compte des ressources limitées des unités mobiles.

Les requêtes effectuées par une UM à un moment donné aussi bien que les réponses aux requêtes requièrent de cette dernière qu'elle réserve de l'espace pour stocker ces informations. Le stockage de ces données peut devenir rapidement prohibitif. Un besoin apparent de définir une stratégie de codage de l'information à stocker pour gagner en espace mémoire s'avère donc indispensable.

Dans les réseaux ad hoc, puisque les UM bougent librement, des ruptures fréquentes de liaisons surviennent et partitionnent involontairement le réseau, empêchant ainsi l'UM d'obtenir rapidement la réponse à sa requête. Par contre, si la requête de l'UM pouvait trouver une réponse auprès d'un voisin disposant d'une copie de l'objet requis, cela lui éviterait de perdre du temps, de la bande passante ainsi que de l'énergie. Hara propose en réponse à ce problème, de mettre en place des mécanismes de réplication de données que nous décrirons dans les prochaines sections [14][15].

De façon générale, la mise en œuvre d'un mécanisme de gestion d'antémémoire dans les réseaux ad hoc est décomposée en deux sous-mécanismes qui constituent chacun, un axe de recherche. Il s'agit de la stratégie à utiliser pour la répartition des données dans le réseau ou encore stratégie de mise en antémémoire, et de la procédure d'invalidation ou de mise à jour de ces données là.

Les caractéristiques intrinsèques des réseaux ad hoc requièrent d'une stratégie de mise en antémémoire la prise en compte des points suivants :

- la limitation en bande passante et en énergie oblige à la définition de mécanismes pour réduire les interactions entre UM et serveur de données ;
- la déconnexion fréquente des UM impose la mise en place de mécanismes efficace pour l'invalidation de l'antémémoire des UM ;
- pour les interactions possibles entre UM et serveur de données, il faudra minimiser la taille des messages à transmettre ;
- la limitation de l'espace de stockage de l'information du côté des UM impose la définition de mécanismes de compression pour sauvegarder de l'espace ;
- dans un réseau ad hoc, la rupture fréquente des liens entre UM oblige à la définition de méthodes pour répliquer des données entre voisins se trouvant à un ou plusieurs sauts.

2.2.2 Stratégies de mise de données en antémémoire

Elles consistent à la mise en place d'une architecture pour la répartition des données dans un réseau tout en minimisant les coûts liés à leur accessibilité. La conception d'une stratégie de mise de données en antémémoire est souvent guidée dans les réseaux filaires par l'hypothèse que le réseau jouit d'une certaine stabilité et d'une bande passante quasiment fixe [16]. On a vu précédemment que ces deux requis entrent en conflits avec les caractéristiques d'un environnement mobile. C'est pourquoi des travaux se sont penchés sur la mise en place de mécanismes plus appropriés pour les réseaux mobiles.

Cooperative cache-based

Pour réduire le délai de réponse aux requêtes et améliorer l'accessibilité aux données, Cao propose des techniques de mise en antémémoire dénommées *CacheData*, *CachePath* et *HybridCache*, utilisant les protocoles de routage sous jacents [12].

CachePath : la Figure 2.3 illustre cette stratégie. En effet, le nœud *N11* représente la source de données tandis que *N1* est un nœud qui gère les transactions de ses voisins à portée radio. *N1* est dit *cluster head.* Supposons que *N1* requiert la donnée d_i de *N11.* Quand *N3* transfert d_i à *N1*, elle sait maintenant que *N1* détient une copie de d_i de sorte que si *N2* en fait la demande prochainement, qu'elle puisse la convoyer à *N1* qui se trouve à un saut, plutôt qu'a *N11* qui se trouve à trois sauts. La connaissance du nombre de sauts d'une UM à une autre est obtenue grâce au protocole de routage (tel que AODV et DSR). La mise en antémémoire du chemin d'une donnée permet ainsi de réduire l'usage de la bande passante ainsi que le délai pour obtenir une réponse à cause du faible nombre de sauts pour accéder à une donnée.

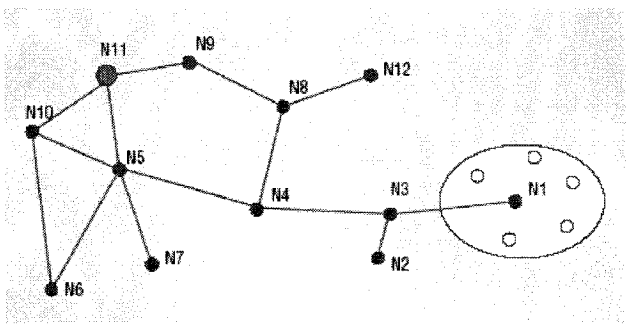


Figure 2.3 Réseau ad hoc et stratégie de mise en antémémoire

Cependant, il importe de noter que l'UM n'enregistre pas les chemins de toutes les données qu'elle transfère. En effet, seuls sont enregistrés les chemins des données qu'elle transfère quand elle se trouve plus proche de l'UM qui requiert la donnée concernée plutôt que de la source de données. Ainsi, quand *N11* transfère la donnée d_i au destinataire *N1* tout au long du chemin *N5-N4-N3*, *N4* et *N5* ne mettront pas en mémoire le chemin correspondant parce qu'ils sont plus proches de la source de donnée que *N1.*

La proximité d'une UM est définie par un ensemble de paramètres fonctions de sa distance à la source de la donnée, sa distance à l'UM mémorisant la donnée, la stabilité de la route ainsi que la fréquence de mise à jour des données.

Soit $H(i,j)$ la distance (en nombre de sauts) de l'UM i à celle j mémorisant la donnée et $H(i,C)$ sa distance à la source de donnée C (possiblement le serveur de données). Le nombre de sauts que *CachePath* peut sauvegarder est de :

$$H_{save} = H(i,C) - H(i,j)$$

Lorsqu'à un moment donné le réseau est stable, c'est à dire avec un faible taux de mise à jour de la source de donnée et $H(i,j) < H(i,C)$, alors l'UM acheminant la donnée peut mettre le chemin afférent dans son antémémoire.

CacheData : l'UM achemine la donnée et ne la met en antémémoire que si elle remarque que cette dernière est fréquemment accédée. À la Figure 2.3, si *N6* et *N7* requièrent la donnée d_i, à travers *N5*, cette dernière se rendra compte que d_i est souvent demandée et la mettra en antémémoire. Des requêtes futures en provenance de *N3*, *N4* ou *N5* pourront ainsi être servies directement par *N5*. Cependant, en supposant que le serveur de données reçoive plusieurs requêtes pour la même donnée d_i transférée par *N3*, toutes les UMs acheminant cette dernière (*N3-N4-N5*) la mettront en antémémoire, croyant qu'elle est populaire. Ainsi, ce phénomène conduit à un gaspillage de l'espace mémoire car chacune de ces UM stocke d_i. Pour éviter cette situation, une UM ne mettra pas en antémémoire une donnée dont la requête est souvent faite par un même requérant. *CachePath* est meilleur quand la taille de l'antémémoire est faible ou lorsque le taux de mise à jour des données est faible, tandis que *CacheData* est meilleur si la taille de la donnée à mémoriser est faible. Pour mieux utiliser leur potentiel, une approche dénommée *HybridCache*, meilleure que les deux précédentes est proposée.

HybridCache : c'est une combinaison des points forts de *CachePath* et de *CacheData*. Quand une UM transfère une donnée, elle met en antémémoire celle-ci ou le chemin pour y accéder en se basant sur des critères tels que sa taille s_i, son TTL_i (*time to*

live) ainsi que le H_{save} précédemment défini. Pour chaque donnée d_i, les opérations suivantes sont effectuées :

- si la taille s_i de la donnée est faible (inférieure à un seuil T_s), *CacheData* est utilisée parce que l'espace requis pour mémoriser la donnée est faible. Dans le cas contraire, *CachePath* est utilisée pour sauvegarder de l'espace ;
- si TTL_i est faible (inférieur à un seuil donné), *CachePath* n'est pas choisi car la donnée risquerait d'être vite invalide et *CacheData* est alors utilisée. Mais, pour un TTL_i élevé, *CachePath* est préféré ;
- si H_{save} est grand, *CachePath* est utilisé, car il permet de sauvegarder un chemin composé de plusieurs sauts. Cependant, lorsqu'une donnée est mémorisée, l'UM prend le soin de mémoriser aussi le chemin pour y accéder ; cela est dû au fait que, si cette donnée venait à être écrasée en raison d'un défaut d'espace, il y aurait toujours un moyen d'y accéder en utilisant le chemin qui y donne accès. La Figure 2.4 illustre l'algorithme qui implémente la méthode *HybrideCache*.

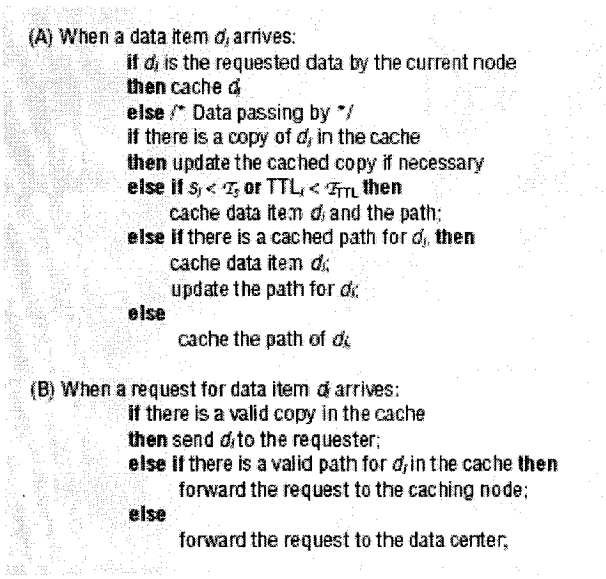


Figure 2.4 Algorithme de la méthode *HybrideCache*

Ces stratégies de mise en antémémoire présentent la faiblesse d'être tributaires du protocole de routage utilisé. Leur efficacité dépendrait de la performance de ce protocole.

Réplication de données

L'accessibilité des données dans un réseau ad hoc est plus faible que dans les réseaux filaires à cause de la topologie dynamique de ces derniers. À la Figure 2.5, la rupture du lien mise en évidence rend inaccessibles pour les UM situées de part et d'autre du lien, les données *D1* et *D2*. Pour résoudre un tel problème dans les réseaux ad hoc, le recours à la réplication de données est une solution possible. En effet, si une UM, d'un côté comme de l'autre, disposait d'une copie de D1 (respectivement D2) comme le montre la Figure 2.6, cette dernière pourrait être toujours accessible et ce, malgré la rupture du lien.

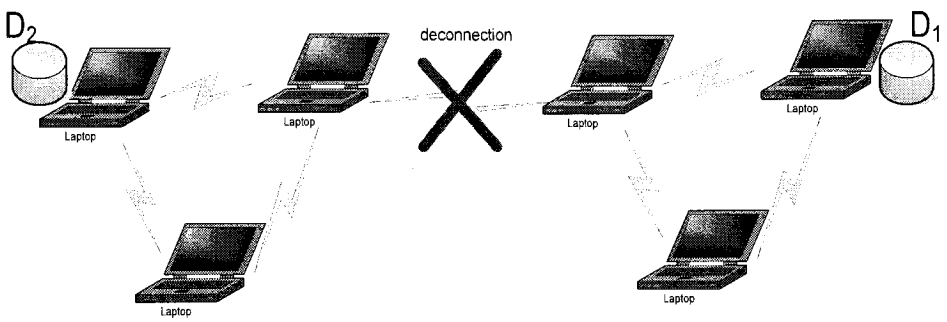


Figure 2.5 Partitionnement par rupture de lien

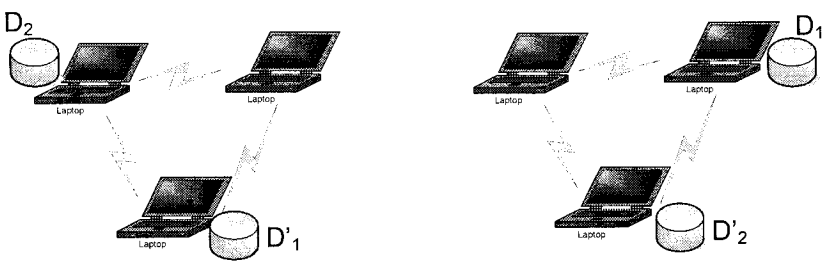


Figure 2.6 Accès aux données par réplication

Pour adapter cette technique aux réseaux ad hoc, Hara propose des stratégies qui assument une mise à jour périodique des données en antémémoire [15]. Quand l'UM effectue une requête sur un objet, cette dernière est un succès (*cache hit*) dans chacun des cas suivants :

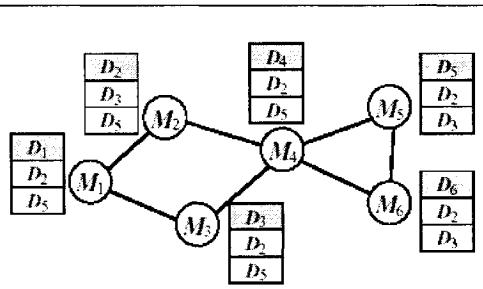
- l'UM possède elle-même l'originale de la donnée ou une copie de cette dernière dans son antémémoire ;
- un voisin se trouvant à un ou plusieurs sauts, dispose de l'original de l'objet requis ou d'une copie.

L'UM vérifie donc l'existence de l'objet dans son antémémoire avant d'effectuer une diffusion de la requête vers ses voisins en cas d'échec. Au cas où elle n'obtient aucune réponse de ses voisins à un ou plusieurs sauts, la requête échoue tout simplement.

Static Access Frequency (SAF) : dans cette stratégie, l'UM effectue une classification des copies d'objets par ordre décroissant de leur fréquence d'accès. Cette méthode présente l'avantage de permettre aux UM de ne plus communiquer une fois que des copies d'objets sont échangées. La Figure 2.7 illustre cette méthode. En effet, on voit à la Figure 2.7 (a) les fréquences d'accès des différents objets D_i détenus par les UM et l'ordre dans lequel chacune d'entre elles les répartit dans son antémémoire.

Data	Mobile host					
	M_1	M_2	M_3	M_4	M_5	M_6
D_1	0.65	0.25	0.17	0.22	0.31	0.24
D_2	0.44	0.62	0.41	0.40	0.42	0.46
D_3	0.35	0.44	0.50	0.25	0.45	0.37
D_4	0.31	0.15	0.10	0.60	0.09	0.10
D_5	0.51	0.41	0.43	0.38	0.71	0.20
D_6	0.08	0.07	0.05	0.15	0.20	0.62
D_7	0.38	0.32	0.37	0.33	0.40	0.32
D_8	0.22	0.33	0.21	0.23	0.24	0.17
D_9	0.18	0.16	0.19	0.17	0.24	0.21
D_{10}	0.09	0.08	0.06	0.11	0.12	0.09

(a) Fréquence d'accès aux données

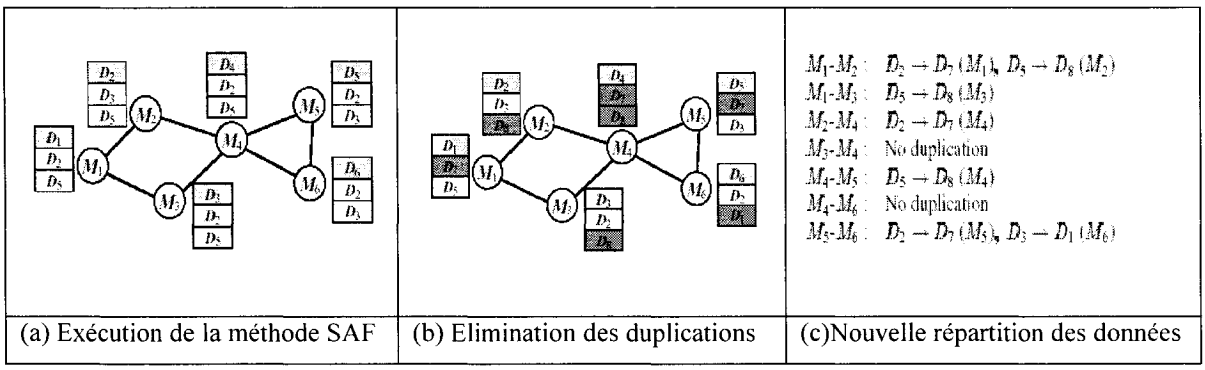


(b) Exemple d'exécution de SAF

Figure 2.7 Exécution de la méthode SAF

L'inconvénient de cette méthode est qu'il existe un risque de duplication d'une copie chez des UM voisines. Il faudrait donc trouver un moyen pour que des UM voisines n'entretiennent pas les mêmes copies d'objet en antémémoire, ce que se propose de faire la méthode DAFN.

Dynamic Access Frequency and Neighbourhood (DAFN) : cette méthode élimine les copies d'objets détenues par des voisins. Elle se sert d'abord de la méthode SAF, puis s'il existe des duplications de copies entre voisins, celui qui dispose de la plus faible fréquence d'accès à l'objet concerné remplace ce dernier par un autre objet. Cette procédure s'exécute périodiquement et est illustrée à la Figure 2.8.



(a) Exécution de la méthode SAF | (b) Elimination des duplications | (c)Nouvelle répartition des données

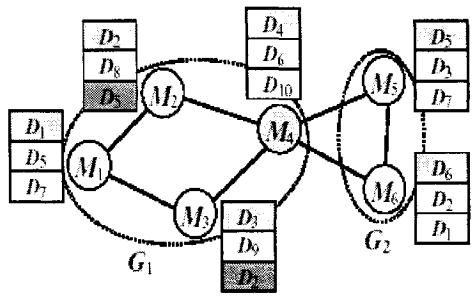
Figure 2.8 Exécution de la méthode DAFN

Cette méthode présente l'inconvénient de l'échange de beaucoup de messages entre voisins et n'élimine pas toujours les duplicas d'objets. D'autre part, si pendant la période de répartition la topologie du réseau change, la procédure s'arrête.

Dynamic Connectivity based Grouping (DCG) : cette méthode partitionne les UM en sous-ensembles connectés les uns aux autres tels des sous-graphes. Même si un sommet du graphe, c'est à dire une UM se déconnecte du sous-ensemble auquel il appartient, ce dernier peut toujours rester connecté aux autres sous-ensembles, d'où une relative garantie de stabilité. La Figure 2.9 illustre son fonctionnement.

Data	Mobile host						Group	
	M_1	M_2	M_3	M_4	M_5	M_6	G_1	G_2
D_1	0.65	0.25	0.17	0.22	0.31	0.24	1.29	0.55
D_2	0.44	0.62	0.41	0.40	0.42	0.46	1.87	0.88
D_3	0.35	0.44	0.50	0.25	0.45	0.37	1.54	0.82
D_4	0.31	0.15	0.10	0.60	0.09	0.10	1.16	0.19
D_5	0.51	0.41	0.43	0.38	0.71	0.20	1.73	0.91
D_6	0.08	0.07	0.05	0.15	0.20	0.62	0.35	0.82
D_7	0.38	0.32	0.37	0.33	0.40	0.32	1.40	0.72
D_8	0.22	0.33	0.21	0.23	0.24	0.17	0.99	0.41
D_9	0.18	0.16	0.19	0.17	0.24	0.21	0.70	0.45
D_{10}	0.09	0.08	0.06	0.11	0.12	0.09	0.34	0.21

(a) Fréquences d'accès des groupes



(b) Exécution de la méthode DCG

Figure 2.9 Exécution de la méthode DCG

En effet, pendant la période de répartition, la procédure DCG calcule la fréquence d'accès du groupe pour un objet donné en additionnant les fréquences d'accès des UM membres du groupe. Si l'original d'un objet existe dans un groupe, la procédure alloue une copie à un membre du groupe. Pour un objet donné, une copie est seulement allouée au membre ayant la fréquence d'accès la plus élevée du groupe. Cette stratégie présente cependant l'inconvénient d'un volume d'échange de messages encore plus important que les deux procédures précédentes.

Pour réduire les délais d'accès et améliorer l'accessibilité aux données, Yin et Cao ont proposé des techniques de réplications proactives de données qui assument que la probabilité d'accès aux objets est connue [17]. Une comparaison de leurs techniques, effectuée avec la méthode DAFN de Hara, a fourni des performances meilleures. Ci-après, nous présentons l'une d'entre elles, dénommée *Greedy-S*.

Greedy-S

Cette stratégie consiste à allouer les données les plus fréquemment accédées à l'espace de stockage jusqu'à ce que ce dernier soit plein. Elle prend en compte la taille de l'objet à stocker et calcule la fréquence d'accès à chaque objet à l'aide de la fonction :

$$F_{ik} = a_{ik} / s_k$$

où a_{ik} représente la fréquence d'accès du noeud N_i à l'objet d_k de taille s_k. Soit C, l'espace réservé au stockage des données. L'algorithme Greedy-S exécuté par chaque nœud est illustré à la Figure 2.10. L'ensemble O représente l'ensemble des objets stockés.

```
Tant que (NombreObjet < C)
{
        Évaluer min{F(O_k)};
        Si (F_ik > min{F(O_k)})
        {
            Détruire l'objet stocké de fréquence min{F(O_k)};
            Ajouter le nouvel Objet O_k ;
            NombreObjet ++ ;
            Servir O_k à l'application requérante ;
        }
        Sinon
        {
            Servir O_k à l'application requérante ;
        }
        Fin Si
}
```

Figure 2.10 Algorithme Greedy-S

Cette stratégie, bien que meilleure à celle DAFN, présente cependant le défaut d'une absence de coopération entre nœuds voisins pour le partage des données. Il ressemble assez à une stratégie de mise en antémémoire dotée d'une politique de renouvellement du type LRU (*Least Recently Used*).

Granularité de l'antémémoire

Chan et al., adressent plutôt la granularité de l'information à stocker en antémémoire, c'est-à-dire son niveau de détail en proposant trois approches de mise de données en antémémoire : *Attribute caching*, *Object caching*, et *Hybrid caching* [18].

Attribute caching : les attributs fréquemment accédés des objets de la base de données du serveur sont mis en antémémoire par l'UM. Après l'évaluation de la requête de l'UM, le serveur retourne à cette dernière uniquement les attributs des objets requis.

Object caching : chaque UM a son propre ensemble d'objets fréquemment accédés et entretenus en antémémoire. En effet, une UM peut accéder à différents attributs d'un même objet dans différentes requêtes. Le serveur, à la réception de telles requêtes, fournit au requérant tous les attributs de l'objet en incluant également les attributs qui ne sont pas requis; ceci permet d'éliminer toute requête future concernant le même objet. L'UM met automatiquement tous les attributs de cet objet en antémémoire.

Hybrid caching : dans la précédente approche, le serveur procure au requérant tous les attributs de l'objet. Pourtant, il est souvent rare que tous les attributs d'un même objet soient accédés à la fois. Cela gaspille non seulement de la bande passante mais aussi l'espace utilisable pour la mémorisation des attributs d'autres objets. C'est dans ce contexte que la stratégie Hybride est utilisée pour permettre plutôt au serveur de données de ne procurer au requérant que les attributs des objets fréquemment accédés. Ces objets sont ceux dont la probabilité d'accès dans le futur est élevée, c'est à dire au delà d'un certain seuil. Ce mécanisme est implémenté par l'usage d'une table dans l'antémémoire de chaque UM pour identifier les attributs et objets s'y trouvant. Sa faiblesse réside dans le fait que chaque UM est supposée communiquer avec un seul serveur de données, ce qui n'est pourtant pas le cas dans les réseaux ad hoc où les usagers passent d'une zone de service de données à une autre.

2.2.3 Mécanismes d'invalidation d'antémémoire

Ils se basent sur l'approche *Stateless Server* et la diffusion de rapports d'invalidation (IR). Le serveur de données diffuse de manière asynchrone ou non un rapport des derniers changements intervenus dans la base de données. Lorsque cette diffusion est asynchrone, les changements sont immédiatement notifiés aux UM. Celles qui à cet instant sont connectées peuvent alors invalider les objets dont elles sont en possession et faisant partir du rapport. Pour éviter que les UM déconnectées perdent tout

le contenu de leur antémémoire, une horodate est ajoutée à chaque objet ayant subi une mise à jour. Ainsi, dès qu'une UM se reconnecte, elle attend le prochain rapport d'invalidation pour mettre à jour son antémémoire. Cette diffusion asynchrone du rapport d'invalidation de l'antémémoire ne garantit aucune borne pour la durée d'attente de l'UM qui s'est reconnectée.

Dans l'approche synchrone, la diffusion de l'IR est périodique. L'UM doit scruter l'IR pour valider ou non les objets se trouvant dans son antémémoire à intervalle de temps régulier. Pour répondre à une requête, l'UM se sert des copies valides d'objets dans son antémémoire. Autrement, elle effectue une requête vers le serveur de données. Ce dernier effectue cette diffusion toutes les L secondes. La Figure 2.11 donne une illustration de l'occurrence des rapports d'invalidation.

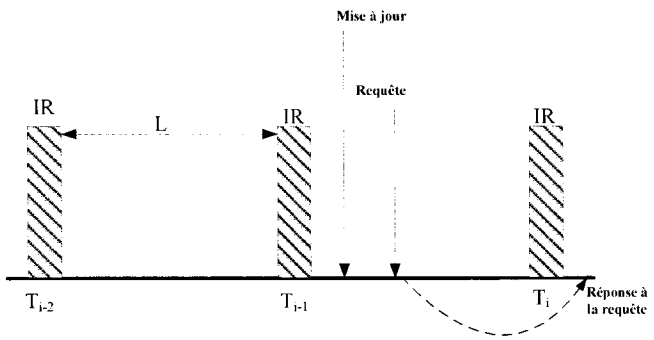


Figure 2.11 Diffusion d'un IR dans le temps

Cao a estimé que si un IR de taille importante rend efficace l'invalidation des antémémoires en procurant plus d'informations sur les changements intervenus dans la base de données du serveur, en revanche il consomme beaucoup de bande passante et oblige l'UM à consommer beaucoup d'énergie [19]. Plusieurs algorithmes d'invalidation d'antémémoire ont été proposés dans la littérature. Nous en présentons ici quelques uns sur lesquels se sont basés la plupart des autres.

Broadcasting Timestamps (TS)

Dans cette stratégie [6], l'IR est composé des horodates des objets qui ont subi un changement dans la base de données durant les w secondes passées. L'UM scrute ce

rapport pour maintenir à jour l'état du contenu de son antémémoire. En d'autres termes, l'IR contient l'historique des mises à jour intervenues pendant les w secondes passées. Pour chaque objet en antémémoire, l'UM le détruit ou non, dépendamment du fait qu'il ait été mis à jour à une date plus récente que celle en antémémoire. À la réception de l'IR, l'horodate de l'antémémoire est mise à jour. Le serveur diffuse toutes les $T_i = i \cdot L$ l'IR et entretient une liste U_i d'objets telle que :

$$U_i = \left\{ [j;t_j] \,/\, \begin{array}{l} j \in D \text{ et} \\ t_j \text{ est l'horodate de la précédente mise à jour de } j \text{ tel que } T_i - w \le t_j \le T_i \end{array} \right\}$$

À la réception de l'IR, l'UM effectue une comparaison avec les objets en antémémoire $[j;t_j^c]$ (ou $j \in D$ et t_j^c représente l'horodate de l'objet j dans l'antémémoire) pour décider si elle garde ou non un objet. L'UM entretient de son coté, une liste Q_i de requêtes telles que :

$$Q_i = \{ j / j \text{ a fait l'objet d'une requête dans l'intervalle de temps } [T_{i-1}, T_i] \}$$

L'UM maintient aussi une variable T_l indiquant l'horodate du dernier IR reçu tel que, si la différence entre l'horodate de l'IR courant et T_l est plus grande que w, alors tout le contenu de l'antémémoire est détruit. Plus particulièrement, l'UM exécute l'algorithme de la Figure 2.12.

Pour sauvegarder de l'énergie, l'UM pourrait ne devenir active que pendant la diffusion de l'IR. Elle peut considérer son antémémoire valide aussi longtemps que la durée de sa déconnexion est plus petite que w.

```
if (T_i − T_l > w) { drop the entire cache }
else {
    for every item j in the MU cache {
        if there is a pair [j, t_j] in U_i {
            if t_j^c < t_j {
                throw j out of the cache }
            else { t_j^c = T_i }
        }
    }
}
for every item j ∈ Q_i {
        if j is in the cache {
            use the cache's value to answer the query }
        else { go uplink with the query }}
T_l := T_i
    }
```

Figure 2.12 Algorithme Broadcasting Timestamps (TS)

Amnesic Terminal (AT)

Dans cette stratégie, le serveur effectue une diffusion des identifiants des objets qui ont connu un changement depuis le dernier IR. Une UM qui entre temps s'est déconnectée, va devoir reconstruire toute son antémémoire. Le serveur construit une liste U_i des objets à diffuser telle que :

$$U_i = \left\{ j \,/\, \begin{array}{l} j \in D \quad \text{et} \\ \text{la précédente mise à jour de } j \text{ est intervenu à } t_j \text{ tel que } \quad T_{i-1} \leq t_j \leq T_i \end{array} \right\}$$

À la réception de cet IR, l'UM compare les objets en antémémoire avec ceux contenus dans l'IR. Si un objet en antémémoire y figure, l'UM le détruit de l'antémémoire, autrement il considère l'objet valide. Elle maintient aussi une liste Q_i des requêtes en instance telle que :

$$Q_i = \{ j / j \text{ fait l'objet d'une requête dans l'intervalle de temps } \quad [T_{i-1}, T_i] \}$$

Par ailleurs, l'UM maintient aussi une variable T_l indiquant l'horodate du dernier IR reçu. Si la différence entre l'horodate de l'IR courant et T_l est plus grande que L, alors tout le contenu de l'antémémoire est détruit. En clair, l'UM exécute l'algorithme de la Figure 2.13.

```
if (T_i − T_l > L) { drop the entire cache }
  else {
    for every item j in the MU cache {
        if j is in the report {
          throw j out of the cache }
    }
  }
  for every item j ∈ Q_i {
      if j is in the cache {
        use the cache's value to answer the query }
      else { go uplink with the query }}
  T_l := T_i
  }
```

Figure 2.13 Algorithme Amnesic Terminal (AT)

La stratégie TS s'avère avantageuse quand la fréquence des requêtes est plus élevée que celle des mises à jour. Quant à la stratégie AT, elle est plus efficace pour les UMs qui se déconnectent rarement.

Update Invalidation Report (UIR)

Dans le but de réduire la latence de la réponse aux requêtes, l'UIR est proposé [19] pour permettre une diffusion de l'IR m fois dans l'intervalle $[T_{i-1}, T_i]$. Le résultat de cette approche pour l'UM qui a une requête en instance est de ne pas attendre un prochain IR avant de répondre. Donc, le délai de latence se voit aussi réduit dans une proportion de *1/m*, par rapport à l'approche **TS**. Puisque l'IR véhicule un volume de données non négligeable, le répliquer m fois consommerait de l'énergie et de la bande passante. Pour pallier ce problème, l'approche propose plutôt d'insérer dans chaque

intervalle $[T_{i-1}, T_i]$, *m-1* mises à jour de l'IR ne contenant que les objets qui ont été mis à jour dans la base de données du serveur depuis la diffusion du dernier IR ; d'où la dénomination de *Update Invalidation Report (UIR)*. La taille de l'UIR devient ainsi plus petite, comparée à celle de l'IR. La Figure 2.14 illustre cette approche.

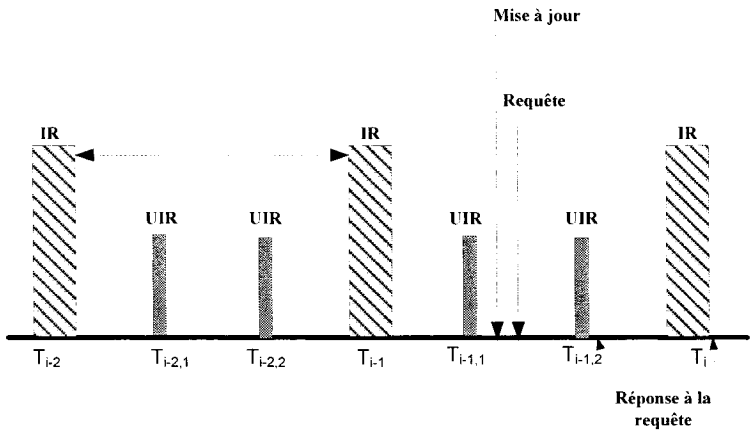


Figure 2.14 Rapport d'invalidation UIR

En effet, quand l'UM reçoit une requête d'un voisin dans l'intervalle de temps $[T_{i-1,1}, T_{i-1,2}]$, elle ne pourra y répondre que dès $T_{i-1,2}$ au lieu de T_i comme dans l'approche **TS**. Sur la base de l'UIR qu'elle reçoit, l'UM peut invalider ou non des objets en antémémoire. S'il existe des copies valides des données requises, l'UM les sert immédiatement à l'usager. Autrement, elle formule une requête en direction du serveur de données.

Puisque le contenu d'un UIR dépend de celui de l'IR précédemment diffusé, chaque UM a besoin de recevoir ce dernier pour utiliser les UIR intermédiaires. En d'autres termes, si une UM manque un IR parce qu'elle était déconnectée pour économiser de l'énergie, elle ne pourra répondre aux requêtes en attente qu'à la réception du prochain IR. La faiblesse de cette approche est qu'elle utilise non seulement beaucoup de bande passante dans le sens descendant, mais aussi ne résout pas le problème des déconnexions.

Bit Sequences

Avec cette stratégie [20], l'IR est composée d'un ensemble de n séquences de bits représentant les mises à jour intervenues aux temps $T_n, T_{n-1}, \ldots, T_1$, avec $T_i < T_{i-1}$ pour $1 \leq i \leq n$. Chaque bit représente un objet de la base de données. Un «1 » représente un objet qui a été mis à jour dans la base de données. Les n séquences binaires sont organisées de manière hiérarchique telle que celle au niveau le plus élevé (séquence B_n) contienne autant de bits qu'il y a d'objets dans la base de données; et celle au plus bas niveau (séquence B_1) contienne 2 bits. Soit l'IR:

$$\{[B_0, T_0], \ldots, [B_k, T_k]\}, \text{ tel que } T_k < T_{k-1}$$

$$B_i = 1 \quad \Leftrightarrow \quad \{\text{La moitié des objets, de 0 a } 2^i \text{ on été mise à jour au temps } T_i\}$$

La séquence B_0 est utilisée avec l'horodate T_0 pour indiquer qu'aucun objet n'a été mis à jour. L'UM utilise la séquence de bits ainsi que l'horodate T_i pour décider quel objet dans son antémémoire doit être invalidé ou non. En clair l'UM exécute l'algorithme décrit à la Figure 2.15.

```
// T – timestamp of current report
// T^c – timestamp of last valid report received by mobile client
if T_0 ≤ T^c
    all cached objects are valid
else {
    if T^c < T_n
        remove the entire cache content
    else {
        determine the bit sequence B_i such that T_i ≤ T^c < T_{i-1}, 1 ≤ i ≤ n
        invalidate all the objects marked "1" in B_i
    }
}
for every object O ∈ Q_i {
    if (O is in the cache)
        use the cache's content to answer the query
    else
        submit request for O
}
T^c ← T
```

Figure 2.15 Algorithme Bit Sequences

Remarque

En général, il existe deux mécanismes de copies de données sur différents sites, dans les systèmes à architecture distribuée comme les réseaux ad hoc [21]. Il s'agit de la mise de données en antémémoire et de la réplication de données. Les deux techniques poursuivent le même but, c'est-à-dire la réduction du coût de communication et de la charge du réseau. Cependant, elles sont différentes sur plusieurs points, comme indiqué au Tableau 2.1.

Tableau 2.1 Différences entre la réplication de données et la mise de données en antémémoire

	Mise de données en antémémoire	Réplication de données
Cibles	client, tierce partie	serveur
Stockage	Mémoire (flash, etc.)	disque
Protocole de mise à jour	invalidation	propagation
Destruction des copies	implicite	explicite
Granularité	fine	brute

La réplication de données bénéficie à un grand nombre d'usagers, alors que la mise de données en antémémoire sert un client ou un groupe restreint de clients. Cependant, elle garantit l'existence de la donnée à l'endroit où on en a le plus besoin. Une combinaison de ces deux techniques dans les réseaux ad hoc améliorerait l'accessibilité aux données.

2.3 Relève du service de données

Dans les réseaux ad hoc, comme les utilisateurs se déplacent d'une zone de service de données à une autre, le serveur de données de la nouvelle zone doit assurer la continuité de l'accès aux données, cela afin de réduire les coûts de communication en bande passante, en énergie et en délai. Ce procédé est désigné sous le nom de relève du service de données (*service handoff*). Pour permettre cette relève du service de données, il existe une technique adaptative dénommée DAR (*Dynamic and Adaptive cache*

Retrieval scheme) utilisant des méthodes appropriées de mise en antémémoire fondées sur l'architecture décrite à la Figure 2.16 [22]. Ces méthodes sont : le FLB, le FPS, et le FCB.

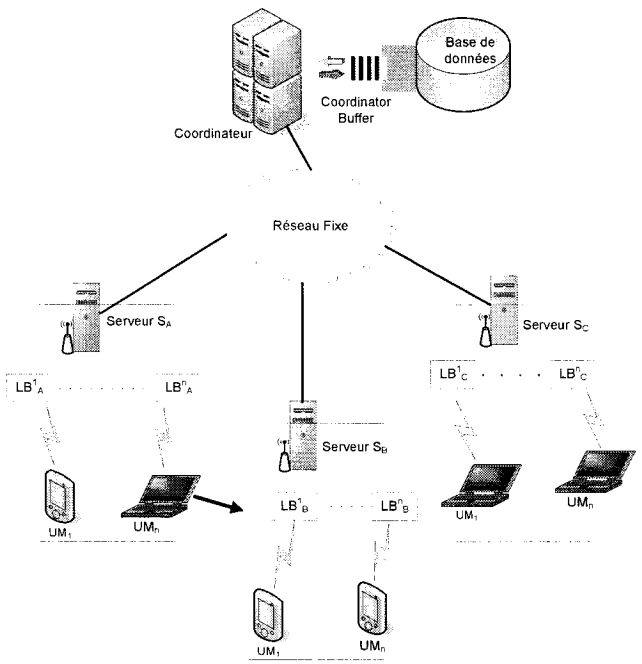


Figure 2.16 Architecture d'une relève du service de données

FLB (Caching from Local Buffer) : due à la propriété asymétrique de la communication entre UM et serveur, ce dernier crée un processus pour chaque UM, de manière à entretenir une instance d'antémémoire locale LB (*Local Buffer*) propre à chaque UM. Le serveur y mémorise les données fréquemment accédées par l'UM. Lorsque l'UM effectue une prochaine requête, elle pourrait ainsi être servie depuis cette antémémoire.

FPS (Caching from Previous Server): pour les UMs effectuant des relèves fréquentes, cette technique est utilisée pour rechercher les données utilisées par l'UM dans le LB maintenu par le précédent serveur de données. Ceci permet de réduire de

façon substantielle la latence d'accès à l'information requise quand l'UM effectue une relève. Cette méthode élimine le coût d'une requête vers la base de données.

FCB (Caching from Coordinator Buffer) : pour les données fréquemment mises à jour dans la base de données et qui sont aussi fréquemment accédées par les UM, cette méthode est utilisée. Elle permet d'aller directement chercher l'information requise par l'UM dans le *Coordinator Buffer*, car elle assume que celles entretenues dans les LB ne sont pas valides. Cela diminue ainsi le délai d'accès.

La méthode DAR quant à elle, est une méthode adaptative qui combine l'efficacité des trois précédentes pour assurer l'accès aux données durant la relève de l'UM. Elle s'exécute en trois phases :

- phase initiale : les méthodes candidates utilisées dans cette phase sont le FLB et le FCB. Lorsque l'UM effectue une requête, la méthode FLB est d'abord utilisée puis en cas d'échec, on recourt à la méthode FCB ;
- phase d'exécution : elle intervient lors de la procédure de relève. Les méthodes candidates sont le FPS, le FCB et le FLB ;
- phase terminale : elle active une procédure d'invalidation des données obsolètes dans les différentes antémémoires.

Ces méthodes sont très attrayantes à cause de la hiérarchie d'exécution des différentes procédures. Cependant, elles souffrent toujours du fait que toutes les UM sont supposées se trouver à un saut du serveur qui dessert leur cellule. La charge de trafic peut facilement être importante au niveau du serveur si les UM sont nombreuses dans une cellule donnée. Il faudrait donc un mécanisme pour réguler cette charge.

2.4 Localisation d'antémémoires

Malgré l'usage d'antémémoires dans les réseaux ad hoc pour diminuer les délais de réponse aux requêtes, l'usage de la bande passante ainsi que la consommation d'énergie, il y a toujours trop de messages transmis et beaucoup de dépenses en énergie. C'est pourquoi Nuggehalli et al., proposent une approche qui consisterait à ne doter

d'antémémoires qu'un nombre limité d'UM [10]. Le problème ainsi posé revient à la localisation des antémémoires dans un réseau ad hoc.

Modélisation du problème

Le réseau ad hoc est modélisé par un graphe $G(V,E)$ où l'ensemble V des sommets représente les UM liées entre elles par l'ensemble des liens E sans fil. Le serveur de données dispose d'un ensemble de données I qui changent toutes les T unités de temps et qu'il peut disséminer aux nœuds via des routes à un ou plusieurs sauts. Pendant l'intervalle de temps T, un nœud quelconque désire la donnée i avec une probabilité p_k tel que $1 \leq k \leq |V|$.

Pour réduire la latence d'accès à la donnée i, au début de chaque intervalle T, le serveur produit i à seulement quelques nœuds du réseau. Cette phase est appelée *phase de dissémination.*

Soit le vecteur $Z = (z_1, z_2, \ldots, z_e)$ où

$$z_e = \begin{cases} 1 & \text{si une copie de } i \text{ est transmise à travers le lien } e \in E \text{ pendant la phase de dissémination} \\ 0 & \text{autrement} \end{cases}$$

À la fin de la phase de dissémination, les nœuds désirant la donnée i, y accèdent avec un coût d'accès exprimé en énergie consommée: C'est la *phase d'accès*.

Hypothèses du modèle

i. La topologie du réseau est considérée stable à un moment donné.
ii. Les liens sont supposés bidirectionnels.
iii. Le serveur possède toujours l'original de la donnée i.
iv. L'énergie requise pour transmettre et pour recevoir la donnée i à travers un lien est supposée constante et égale à 1.
v. Le coût de la latence d'accès pour tout nœud k est modélisé par le nombre minimum de sauts pour atteindre la donnée i.
vi. Le coût de stockage de la donnée i est supposé négligeable.

Calcul des coûts

Soient :

C l'ensemble des nœuds désignés pour disposer d'une antémémoire.

d_k, la distance minimale en nombre de sauts de n'importe quel nœud k à C.

Le coût de l'énergie consommée pendant la phase de dissémination de i est donnée par :

$$K_{diss-energy} = \sum_{e \in E} z_e \ .$$

Le coût de l'énergie consommée pendant la phase d'accès assimilé également au coût de la latence d'accès à la donnée i par un nœud k est donné par :

$$K_{access-energy} = K_{latency} = \sum_{k \in V} p_k d_k$$

$$K_{energy} = K_{access-energy} + K_{diss-energy} \ .$$

Le coût total est donné par :

$$K_{tot} = K_{energy} + \lambda \cdot K_{latency} = \sum_{e \in E} z_e + (1+\lambda) \cdot \sum_{k \in V} p_k d_k$$

où λ indique l'importance relative de la latence d'accès et de l'énergie consommée.

L'objectif du problème est de trouver une stratégie Z qui minimise le coût total, en essayant de trouver un compromis entre l'énergie moyenne consommée et la latence d'accès.

Formulation du problème

La formulation revient à un problème de programmation en nombres entiers dont la fonction objectif est :

$$\text{P : Minimiser} \sum_{k \in V} \sum_{e \in E} p_k c_{ke} x_{ke} + M \sum_{k \in E} z_e \quad \textbf{(2.1)}$$

$$\sum_{e \in E} x_{ke} \geq 1 \qquad \forall k \in V \quad \textbf{(2.2)}$$

$$z_e - x_{ke} \geq 0 \qquad \forall k \in V, e \in E \quad \textbf{(2.3)}$$

$$\sum_{\delta(S)} z_e - \sum_{e \in S} x_{ke} \geq 0 \quad \forall S \subseteq E, e_v \notin S, \forall k \in V \quad \textbf{(2.4)}$$

$$x_{ke}, z_e \in \{0,1\} \qquad \forall k \in V,\ e \in E \quad \textbf{(2.5)}$$

où $M = 1/(1+\lambda)$ et c_{ke} représente le coût de la latence d'accès du nœud k à la donnée i au nœud antémémoire du lien e.

$\{x_{ke}\}$ est l'ensemble des variables d'assignation telles que :

$$x_{ke} = \begin{cases} 1 & \text{si le nœud } k \text{ accède a la donnée i du nœud antémémoire du lien e} \\ 0 & \textit{autrement.} \end{cases}$$

Résoudre (2.1) revient à minimiser l'énergie K_{tot}. La contrainte (2.2) assure que chaque nœud est au moins assigné à un nœud antémémoire à partir duquel il peut accéder à la donnée *i*. La contrainte (2.3) assure que chaque nœud appartient à au moins un lien disposant d'un nœud antémémoire. La contrainte (2.4) est induite par la connectivité du sous-graphe formé par *Z*. La contrainte (2.5) représente celle de cardinalité.

Pour résoudre ce problème NP-complet, afin de déterminer quels nœuds peuvent abriter une antémémoire, une heuristique dénommée *POACH (Power Aware Caching Heuristic)* est proposée pour produire une solution approximative au problème **P**.

Le problème avec cette approche vient du fait que la localisation des nœuds pouvant abriter une antémémoire est dynamique et prendrait plus de temps. La topologie

peut déjà changer avant une nouvelle relocalisation. De même, lorsque le nombre d'UM est élevé, cette approche deviendrait très vite coûteuse en délai de relocalisation.

2.5 Un modèle de mise en antémémoire

Le modèle de mise en antémémoire que nous présentons ici est celui du protocole WAP (*Wireless Application Protocol*) utilisé dans les réseaux cellulaires pour permettre d'apporter aux usagers des services autres que la voix *(e-mail, web browsing)*. Ce modèle est présenté, car il correspond à un exemple typique de l'utilisation d'antémémoires dans les réseaux mobiles [23]. Pour rappel, la Figure 2.17 décrit l'architecture d'un réseau utilisant ce protocole.

En effet, le mécanisme de gestion d'antémémoires utilisé par le protocole WAP est dénommé *Cache Operation*. Il offre des moyens d'invalider les objets maintenus en antémémoire par l'agent de l'usager, instance de gestion de l'antémémoire dans le terminal de l'usager. Son contenu est spécifié sous forme d'un document XML dont la structure est faite d'URIs (*Uniform Ressource Identifier*) d'objets à invalider. Deux types d'opérations sont effectués :

- *invalidate object* : permettant d'invalider un objet identifié par une URI donnée ;
- *invalidate service* : permettant d'invalider tous les objets utilisant le même préfixe d'URI.

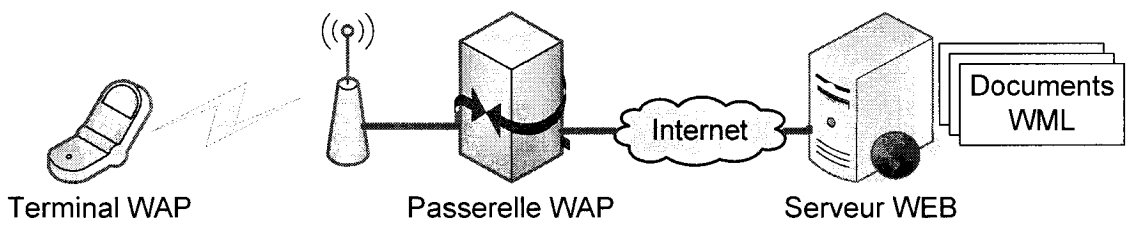


Figure 2.17 Architecture d'un réseau WAP

Cache Operation requiert de l'agent de l'usager une invalidation d'un ou de plusieurs objets dès réception du message CO. Cette opération d'invalidation peut être initiée par le serveur du domaine de l'opérateur ou par l'agent de l'usager. Si aucune

URI du CO n'identifie un objet de l'antémémoire d'une UM, cette dernière détruit tout simplement le CO. Le contenu du CO est encodé grâce à une représentation binaire compacte basée sur le WAP Binary XML utilisant des *Tokenizer* tel que le montre le scénario décrit à la Figure 2.18.

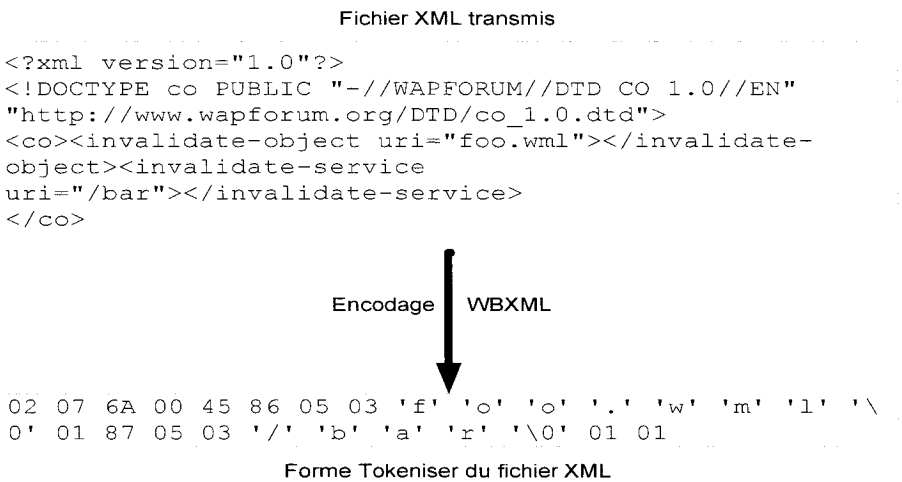


Figure 2.18 Compression de données au format XML avec usage de Tokenizer

En effet, le format textuel qui est constitué de 150 octets se voit réduit grâce aux Tokenizer à 27 octets, soit une réduction de 82% de l'information à transmettre; ce qui est remarquable pour l'usage efficace de la bande passante. Si on imagine qu'à la suite de cette réduction du nombre d'octets existe une procédure de compression, cela pourrait diminuer davantage la quantité d'informations à transmettre sur le médium.

Tous ces mécanismes de mise de données en antémémoire et de réplication de données, offrent des attraits qu'utilisera le modèle de mise de données en antémémoire que nous proposons dans le prochain chapitre.

CHAPITRE III
MODÈLE DE MISE DE DONNÉES EN ANTÉMÉMOIRE

L'objectif ultime de ce mémoire est de proposer une stratégie de mise de données en antémémoire. Nous avons vu dans le chapitre précédent que la bande passante à la disposition des UM est limitée, ainsi que l'espace de stockage dont elles disposent. D'autre part, leur liberté de mouvement fait que les déconnexions sont fréquentes, occasionnant souvent un partitionnement arbitraire de la topologie du réseau qu'elles constituent. Les UM d'une partition ne peuvent donc plus accéder aux données d'une autre partition. L'accessibilité aux données est donc faible, comparativement aux réseaux filaires. Deux techniques attractives telles que la mise de données en antémémoire et la réplication de données sont généralement utilisées dans les réseaux filaires pour améliorer l'accessibilité aux données. Leur application aux réseaux ad hoc constitue un ensemble de défis pour les raisons précédemment énumérées. Dans ce chapitre, nous proposons un modèle pour la mise de données en antémémoire, en tenant compte des contraintes auxquelles sont soumises les unités mobiles. De façon spécifique, ce modèle utilisera une combinaison des techniques de réplication et de mise en antémémoire vues dans le précédent chapitre. Nous le formulons sous la forme d'un problème d'optimisation et proposons une approche pour sa résolution. Viendra par la suite une esquisse de l'approche de résolution sous la forme d'algorithmes à mettre en œuvre au sein d'agents au niveau du serveur et des UM.

3.1 Analyse du problème

Plusieurs aspects nécessitent d'être pris en compte pour la définition d'une solution aux problèmes de la mise de données en antémémoire.

La limitation en bande passante et en énergie oblige à la définition de mécanismes pour réduire les interactions entre UM et serveur de données. Il faudra aussi pour les

interactions nécessaires définir une méthode pour réduire la taille des messages échangés.

La restriction de l'espace de stockage de l'information du côté des UM impose la définition de mécanismes de compression de données pour sauvegarder de l'espace. Rappelons que les techniques de mise en antémémoire utilisées actuellement que ce soit dans les réseaux filaires qu'ad hoc, tiennent compte du fait que chaque entité dispose d'une mémoire pour le stockage des données auxquelles elle accède fréquemment. La conséquence dans une telle situation est le gaspillage de l'espace, car les mêmes informations sont stockées au niveau de plusieurs UM. Mais cela garantit toutefois des coûts d'accès faibles aux objets. Dans un tel contexte, il faut définir un mécanisme qui permette de spécifier les UM devant disposer d'une antémémoire ou d'un réplica des données du serveur. On parlera plutôt de *quasi-réplica* (QR), car les UM ne peuvent héberger toute la base de données du serveur.

La rupture fréquente des liens entre UM ainsi que le partitionnement arbitraire du réseau rallonge le délai d'accès à l'information désirée. Il faudra, pour réduire cette latence d'accès, définir une méthode pour minimiser la distance entre l'UM et son serveur de données. De même, le dynamisme de la topologie du réseau impose la mise en œuvre de mécanisme de mise à jour des données liées à la mobilité des UM.

Pour tenir compte de tous ces requis, le modèle que nous formulons plus loin, met un accent particulier sur la combinaison des techniques de réplication de données et de mise en antémémoires. L'ensemble de ces requis est mis sous forme d'un problème d'optimisation à la fois de l'espace de stockage, du délai d'accès aux données, ainsi que de celui du choix des UM pouvant héberger des réplicas de données du serveur.

3.2 Formulation du modèle

Le problème que nous allons tenter de résoudre peut être reformulé de la façon suivante : « *Comment choisir des unités mobiles clés dans un réseau ad hoc pour y répliquer des données localisées sur un serveur distant ? Comment augmenter le pourcentage d'accès à l'information tout en réduisant le délai d'accès à celle-ci ?* »

Le partitionnement des réseaux ad hoc permet une flexibilité de leur contrôle ainsi que de leur gestion ; c'est-à-dire des méthodes de routage de l'information, de l'allocation des ressources ainsi que de la mise à jour de leur topologie [24]. Ce partitionnement peut être effectué de plusieurs façons pour couvrir tout le réseau. Les avantages offerts par le découpage en partitions (*cluster* en anglais) sont notamment :

1. une réutilisation spatiale des ressources à disposition du réseau;
2. une réduction du nombre de messages de routage propagés à travers le réseau ; en effet, chaque cluster dispose d'un leader (*cluster head*) qui coordonne le trafic des membres (*cluster member*);
3. une vue globale d'un cluster à travers son leader est généralement suffisante pour une prise de décision de routage de l'information ;
4. en particulier, la mise à jour de la topologie du réseau peut se faire de façon isolée entre clusters, ou entre leaders.

Pour bénéficier de tels avantages, nous proposons une approche basée sur le partitionnement du réseau. Dans notre modèle, cela consistera à désigner des UM clés pour héberger des partitions de données du serveur. Nous identifierons ces UM par la dénomination de *Quasi-Réplica* (QR). Elles représentent les *clusters head*. Puis, d'autres UM s'associeront à ces QR dépendamment de leur proximité, pour accéder aux données entreposées. Celles-ci sont des *clusters member*.

Un aspect important de la solution que nous proposons est la combinaison de la réplication partielle de données du serveur avec une technique de mise en antémémoire. Ainsi, les UM désignées QR ne disposent plus d'un espace de stockage dédié pour l'antémémoire, contrairement aux autres non-QR qui, elles, en disposent. Cet espace dédié pour l'antémémoire est muni d'une politique de renouvellement des objets reçus par l'UM lors de ses diverses requêtes.

3.3 Localisation des Quasi-Réplica

Dans cette section, nous introduisons plus en détail le concept de *quasi réplica* (*QR*). Puisqu'une UM ne peut stocker entièrement le contenu de la base de données du

serveur, on y entreposera les données les plus fréquemment accédées, dépendamment de son espace de stockage: on parlera alors de *réplication partielle*.

Ainsi, la zone de couverture du serveur de données est découpée en clusters, gérées chacune par un QR. Ce dernier est désigné par le serveur sur la base de critères tels que la vitesse, l'espace de stockage, l'énergie à disposition ainsi que la densité (nombre de voisins) des UM. Une définition analytique de ces terminologies est présentée dans les prochaines sections.

L'objectif du modèle est de rapprocher les QR le plus possible des UM, pour réduire les coûts d'accès aux données (typiquement la distance entre UM et QR). Nous désignerons par la suite, le serveur de données par MSS (*Mobile Support Station*) comme dans la littérature.

En effet, bien que le MSS soit capable d'atteindre toutes les UM se trouvant à sa portée radio en un seul saut, il n'en est pas de même des UM dont la portée de transmission est moins importante.

Le fait de partitionner chaque zone de service de données en cluster offre les avantages que nous avons cités plus haut, mais génère un coût à payer quant à la maintenance des clusters. En effet, lorsqu'un QR devient indisponible pour une raison ou une autre, il faudrait trouver une alternative pour la continuité du service de données. De même, quand un cluster se retrouve isolé, il importe également de définir un moyen de combler la faille engendrée par son absence.

Dans notre architecture, afin d'éviter ces procédures de maintenance qui contribueraient à inonder le réseau de messages supplémentaires, nous adoptons une approche basée sur le choix de la localisation de QR, afin de permettre aux UM d'effectuer leurs requêtes auprès du QR le plus proche en terme de distance.

Une illustration de l'architecture proposée est faite à la Figure 3.1. En effet, on y voit sous forme de bâtons cylindriques une représentation des UM désignées QR, ainsi que des petits points tout autour de ces bâtons représentant les UM associées à un moment donné au QR. Il faut mentionner que les QR représentent les UM sur lesquelles le serveur fait une réplication partielle de ses données.

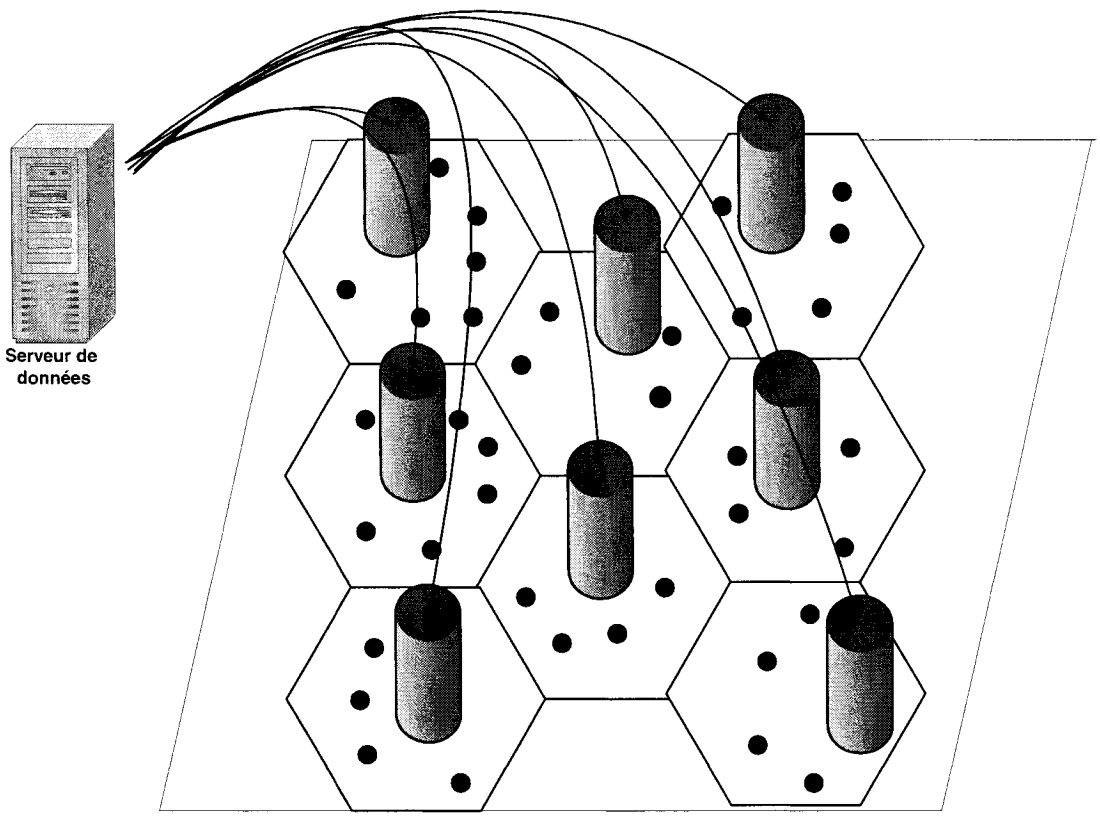


Figure 3.1 Architecture de désignation des QR

La délimitation des cellules ou cluster est dynamique dans le temps, c'est à dire qu'une UM peut acheminer ses requêtes au QR de son choix dépendamment de sa proximité. En effet, chaque UM non désignée QR dispose d'une table répertoriant la liste des QR désignés par le MSS. Cette liste est diffusée par ce dernier, à l'issue de la procédure de désignation.

3.3.1 Description analytique du modèle

La désignation judicieuse des QR requiert la prise en compte de paramètres tels que la vitesse moyenne des UM, leur degré de connectivité, l'énergie instantanée dont elles disposent, leur capacité de stockage ainsi que la popularité des données entreposées dans la base de données du MSS. Il faut donc, sur la base de ces paramètres, définir un

profil d'une UM éligible. Avant de détailler la procédure de désignation des QR, nous introduirons quelques terminologies qui serviront à l'élaboration du modèle.

Vitesse moyenne d'un nœud

Chaque nœud calcule sa vitesse moyenne, grandeur donnant une idée de sa mobilité dans le temps de la façon suivante :

$$M_v = \frac{1}{T}\sum_{t=1}^{T}\sqrt{(X_t - X_{t-1})^2 + (Y_t - Y_{t-1})^2 + (Z_t - Z_{t-1})^2} \qquad \textbf{(3.1)}$$

Les variables X_t, Y_t et Z_t représentent les coordonnées des différentes positions occupées par l'UM au cours du temps tandis que T représente l'instant auquel cette vitesse est estimée.

Degré de connectivité

Le degré de connectivité d'une UM v représente le nombre d'UM dans son voisinage $N(v)$, c'est-à-dire l'ensemble des autres UM se trouvant à sa portée radio. Elle est définie de la façon suivante :

$$D_j = |N(j)| = \sum_{j' \in V, j \neq j'} \{dist(j, j') < R\} \qquad \textbf{(3.2)}$$

Énergie de la batterie

L'énergie E_v à la disposition d'un nœud à un instant T donné est exprimée par :

$$E_j = E_0 - \frac{1}{T}\sum_{t=1}^{T}(E_t - E_{t-1}) \qquad \textbf{(3.3)}$$

où E_0 représente l'énergie intrinsèque de l'UM, et E_t les différentes valeurs de l'énergie au cours du temps.

Loi de Zipf

Dus à l'explosion d'Internet, les serveurs d'antémémoire connus sous la dénomination de *web proxy caching servers* sont utilisés comme des moyens d'améliorer l'accessibilité aux pages Web de serveurs distants. Plusieurs recherches ont

été effectuées pour modéliser le comportement des requêtes effectuées sur les pages Web par les usagers d'Internet [8]. Ces recherches ont démontré que les requêtes effectuées par les usagers des pages Web suivent une distribution particulière dénommée *distribution comportementale de Zipf* (*Zipf-Like distribution* en anglais).

La distribution de Zipf définit la popularité d'un objet, et statue que la probabilité d'accès au *k-ième* objet d'un serveur ($1 \leq k \leq N$) est inversement proportionnelle au rang de cet objet, telle que :

$$p_k = \frac{1}{k^{\theta} \sum_{i=1}^{N} \frac{1}{i^{\theta}}} \qquad \textbf{(3.4)}$$

où $0 \leq \theta \leq 1$. Pour $\theta = 0$, la distribution devient uniforme et pour $\theta = 1$, elle devient une distribution stricte de Zipf. Pour de grandes valeurs de θ, les accès se focalisent sur les objets de faible rang.

3.3.2 Le modèle

Les hypothèses régissant le modèle sont les suivantes :

1. Le serveur de données MSS est considéré comme une référence géographique par les UM se trouvant dans sa zone de couverture. En effet la plupart des méthodes de localisation utilisent le GPS (*Global Positionning System*) dont la précision n'est pourtant pas efficace, spécialement lorsque des entités équipées de récepteur GPS se retrouvent dans un espace clos. Cependant, il existe aujourd'hui une méthode plus adaptée pour les réseaux sans fil utilisant la technologie SDMA (*Space Division Multiple Access*) dans laquelle une UM dotée d'antennes intelligentes (*smart Antenna*) focalise sa transmission vers d'autres UM de son choix. Cette nouvelle technologie utilisée de plus en plus dans les réseaux sans fil est combinée avec la localisation GPS pour améliorer la précision de la localisation du récepteur. Nous ferons ainsi l'hypothèse que la couche physique des différentes entités du réseau est dotée d'un tel équipement muni d'un récepteur GPS. Notons que dans la littérature existent des protocoles de couche MAC pour ce type de support physique [25]. Le

MSS diffuse donc l'information sur sa position, et chaque UM requérante est donc capable de lui fournir les informations telles que sa capacité de stockage S_j, ses coordonnées géographiques (X_j, Y_j, Z_j) relatives au MSS, sa vitesse moyenne M_j, l'énergie à disposition de sa batterie E_j, ainsi que son degré de connectivité D_j. Ces informations sont transmises au serveur de données lors des différentes requêtes effectuées par l'UM à travers un ou plusieurs sauts ;

2. Le MSS possède en tout temps une copie de la donnée requise par une UM, dans sa base de données. Il dispose d'une puissance de calcul largement supérieure à celle des UM ainsi qu'une portée radio conséquente pour alimenter les UM se trouvant dans sa zone de couverture ;
3. Contrairement à l'architecture proposée dans la littérature et décrite dans le chapitre précédent qui assume que le serveur est fixe, nous ferons plutôt dans notre modèle l'hypothèse qu'il est doté d'une mobilité réduite. Il n'est donc pas forcément stationnaire ;
4. Les mécanismes d'invalidation d'antémémoire ne sont pas pris en compte dans ce modèle, ainsi que la gestion du handoff pour une UM passant d'une zone de service de données à une autre (d'un MSS à un autre). La gestion du mécanisme du handoff pour le service de données est déjà couverte par des entités supérieures telles que les routeurs d'accès (compatibles Mobile IP). On considère ainsi que le MSS se trouve dans un domaine couvert par un routeur d'accès.

Notation

La notation suivante est utilisée pour la modélisation. Soient :

N l'ensemble des nœuds se trouvant dans la zone de couverture du MSS et D l'ensemble des objets entreposés dans la base de données du MSS;

H l'ensemble des UM élues pour abriter un QR et E l'ensemble des états possibles d'une UM.

z_j variable 0-1 telle que :

$$z_j = \begin{cases} 1 & \text{si un QR est installé sur l'UM}_j \\ 0 & \text{sinon} \end{cases}$$

x_{ij} variable 0-1 telle que :

$$x_{ij} = \begin{cases} 1 & \text{si l'UM}_i \text{ accède à un objet dans le QR installé sur l'UM}_j \\ 0 & \text{sinon} \end{cases}$$

y_{jk} variable 0-1 telle que :

$$y_{jk} = \begin{cases} 1 & \text{si l'objet } O_k \text{ est entreposé au QR installé sur l'UM}_j \\ 0 & \text{sinon} \end{cases}$$

C_k, le coût du stockage de l'objet O_k au QR localisé sur l'UM$_j$ et U_j le coût lié au choix d'une UM pour héberger un QR; c_{ij} le coût d'accès de l'UM$_i$ à un objet entreposé au QR de l'UM$_j$.

F_c la fonction de compression des objets stockés dans les QR; α, β borne inférieure (resp. supérieure) de la taille d'un cluster géré par un QR préalablement désigné; les variables η_1 et η_2 désignent la borne inférieure (respectivement supérieure) du nombre de QR qu'il pourrait y avoir dans le réseau en tout temps.

Les variables *m, n, t, q, r, h* sont des facteurs de coût, exprimé en $\$/_{unité}$. La procédure de désignation des QR suit la fonction de coût F_{QR} illustrée à la Figure 3.2. Cette fonction minimise : le coût lié au choix des UM devant héberger un QR, le coût lié au stockage des données et enfin le coût d'accès des UM à l'information requise.

$$\textbf{Minimiser } F_{QR} = \sum_{i\in N\setminus H}\sum_{j\in H} c_{ij}x_{ij} + \sum_{j\in H}\sum_{k\in D} C_k y_{jk} + \sum_{j\in N} U_j z_j \qquad \textbf{(3.5)}$$

$$\text{Sujet à :} \quad \alpha \le \sum_{i\in N\setminus H} x_{ij} \le \beta \quad (j\in H) \qquad \textbf{(3.6)}$$

$$\sum_{j\in H} x_{ij} = 1 \quad (i\in N\setminus H) \qquad \textbf{(3.7)}$$

$$\sum_{k\in D} F_c(O_k) y_{jk} \le S_j \quad (j\in H) \qquad \textbf{(3.8)}$$

$$\left.\begin{array}{l} D_j \ge \overline{D} \\ E_j \ge \overline{E} \\ M_j \le \overline{M} \\ S_j \ge \overline{S} \end{array}\right\} \quad (j\in N) \qquad \textbf{(3.9)}$$

$$\eta_1 \le \sum_{j\in N} z_j \le \eta_2 \qquad \textbf{(3.10)}$$

Figure 3.2 Fonction de coût d'une procédure de désignation de QR

La contrainte (3.6) permet de délimiter le nombre maximum d'UM pouvant effectuer des requêtes sur le QR placé sur l'UM j. Cette contrainte est nécessaire pour permettre de répartir la charge du trafic de façon quasi uniforme sur l'ensemble des QR désignés. La contrainte (3.7) indique qu'une UM ne peut effectuer à un instant donné ses requêtes qu'auprès d'un et un seul QR. La contrainte (3.8) quant à elle, permet de limiter le nombre d'objets du MSS stockables sur le QR au moyen de la fonction de compression de données F_c . En effet, l'usage d'un algorithme de compression permettrait un stockage plus important d'objets sur les QR désignés. La contrainte (3.9) elle, définit le profil d'une UM susceptible d'être élue QR. On l'appelle *le profil d'éligibilité*. La contrainte (3.10) définit le nombre de QR éligibles.

Dans le profil d'éligibilité, une UM dispose d'une vitesse moyenne en dessous de la moyenne des vitesses de l'ensemble des UM, une capacité de stockage au dessus de la

moyenne, une énergie plus importante que la moyenne et enfin un degré de connectivité supérieur à la moyenne. Toutes ces variables réunies sous forme de combinaison linéaire, définissent la fonction de coût U_j liée au choix d'une UM. Cette fonction de coût est définie comme une pondération des variables du profil telle que :

$$U_j = m \cdot M_j + n \cdot D_j + t \cdot E_j + q \cdot S_j \qquad m+n+t+q=1 \qquad \textbf{(3.11)}$$

L'hébergement d'un QR à vide (ne stockant aucun objet) par une UM coûte moins cher si cette dernière a une faible mobilité dans le temps, possède beaucoup de voisins et dispose de plus d'énergie que la moyenne. Le coût d'accès c_{ij} de l'UM i au QR j est proportionnel à la distance entre les deux entités, soit :

$$c_{ij} = h\sqrt{(X_j - X_i)^2 + (Y_j - Y_i)^2 + (Z_j - Z_i)^2} \qquad \textbf{(3.12)}$$

Le coût de stockage d'un objet est proportionnel à la taille de l'objet rendue par la fonction de compression F_c ainsi qu'à sa probabilité d'accès p_k, telle que :

$$C_k = r \cdot (1 - p_k) \cdot F_c(O_k) \qquad \textbf{(3.13)}$$

Dans cette fonction de coût, un objet fort populaire coûtera plus cher à stocker qu'un autre à faible popularité, ceci à taille égale. Le nombre d'états à examiner pour désigner les UM accueillant des QR est :

$$|E| = 2^{|N|} \qquad \textbf{(3.14)}$$

La résolution du modèle peut devenir très coûteuse en temps de calcul (NP-difficile) si le nombre d'usagers dans une zone de service de données est élevé. Le modèle ainsi formulé est proche de celui de la localisation de commutateurs dans un réseau filaire soluble en général à l'aide d'une heuristique hors ligne comme la recherche taboue.

Dans notre cas, c'est-à-dire avec les réseaux ad hoc, trouver une solution à l'aide d'une heuristique en ligne telle que la recherche taboue sans que la topologie n'ait changé, puis, charger les QR d'objets populaires relèverait d'un exercice trop coûteux en temps de calcul.

Il est donc peu probable d'utiliser une méthode approximative en ligne pour garantir une bonne solution dans des délais bornés. Donc, nous utiliserons une approche adaptative, c'est-à-dire une méthode qui inclut la coopération de toutes les entités (MSS, UM et QR). Pour faciliter la mise en œuvre de cette méthode de localisation des QR, nous allons décomposer la résolution du problème en trois phases :

Phase I : Élection des QR

Durant cette phase, le profil d'éligibilité est utilisé pour évaluer la valeur de la fonction objectif, donnée par :

$$\textbf{Minimiser} \sum_{j \in N} U_j z_j \quad \textbf{Sujet à :} \left\{ \begin{array}{l} D_j \geq \overline{D} \\ E_j \geq \overline{E} \\ M_j \leq \overline{M} \\ S_j > \overline{S} \end{array} \right\} \textbf{et} \quad \eta_1 \leq \sum_{j \in N} z_j \leq \eta_2 \qquad \textbf{(3.15)}$$

De cette évaluation est obtenu l'ensemble *H* des UM élues pour héberger un QR.

Phase II : Chargement des QR

C'est la phase d'approvisionnement des QR en objets entreposés dans la base de données du MSS. En effet, nous avons formulé que l'assignation d'objets du MSS aux différents QR désignés est donnée par la fonction objectif :

$$\textbf{Minimiser} \sum_{j \in H} \sum_{k \in D} C_k y_{jk} \quad \textbf{Sujet à :} \sum_{k \in D} F_c(O_k) y_{jk} \leq S_j \quad (j \in H) \qquad \textbf{(3.16)}$$

Cette fonction objectif essaye de minimiser le coût d'affectation des objets du MSS aux QR. Dans un tel contexte, un objet peut se retrouver assigné à un QR sans pour autant être populaire dans la région desservie par ce dernier. De la même façon, un QR peut se retrouver uniquement chargé d'objets qui n'ont aucun intérêt pour sa région.

Par ailleurs, ce problème d'affectation n'est pas soluble de manière exacte lorsque la taille de la base de données du MSS est importante. L'usage d'une méthode approximative pour délivrer une solution requiert un temps de calcul non borné, alors que la topologie du réseau est dynamique.

Pour contourner ces limitations du modèle, nous utiliserons plutôt une approche qui consisterait à évaluer pour chaque QR désigné, en fonction de son espace de stockage S_{jQR} disponible, le nombre d'objets populaires qu'il peut héberger.

Comment choisir donc les k objets les plus populaires de manière à garantir un pourcentage de succès élevé pour l'accès à l'information requise ?

La réponse à cette question, c'est-à-dire la valeur de k déterminera le nombre d'objets que le MSS devra charger dans chaque QR.

Soit $D = \{O_i \mid 1 \le i \le M\}$ l'ensemble des objets entreposés sur le MSS et accédés par l'ensemble des UM durant un intervalle de temps Δt. En effet, pour suivre l'évolution de la dynamique du réseau, le temps est découpé en intervalles égaux au cours desquels le MSS reçoit les requêtes des usagers. La Figure 3.3 illustre cette situation.

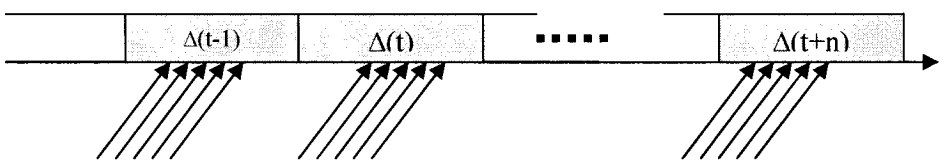


Figure 3.3 Occurrence des requêtes dans le temps

La popularité des M objets entreposés sur le MSS est connue grâce à la loi de Zipf. Pour ce faire, déterminons le nombre d'accès aux k objets les plus populaires :

Soit N_a le nombre d'accès concentrés sur l'ensemble des M objets du MSS. Le nombre d'accès à l'objet O_i est :

$$N_i = N_a \cdot P_i$$

où P_i est la probabilité d'accès à l'objet O_i. Le nombre total d'accès aux k objets les plus populaires est donc de:

$$N_{TOT} = \sum_{i=1}^{k} N_i = N_a \cdot \sum_{i=1}^{k} P_i$$

Dans l'expression de la distribution de zipf :

$$P_k = \frac{1}{k^{\theta} \sum_{i=1}^{M} \frac{1}{i^{\theta}}}$$

Posons :

$$a = \frac{1}{\sum_{i=1}^{M} \frac{1}{i^{\theta}}} \quad \Rightarrow \quad P_k = \frac{a}{k^{\theta}}$$

Faisant l'approximation que θ est proche de 1 ($\theta \approx 1$), on observe que :

$$a = \frac{1}{H_M} \quad \textit{où } H_M \textit{ est le nombre harmonique d'ordre } M \textit{ tel que:}$$

$$H_M = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots\cdots + \frac{1}{M} \cong \ln M$$

La probabilité d'accès à l'objet de rang k devient : $P_k = \frac{1}{H_M}$;

L'expression du nombre total d'accès sur les k objets les plus populaires devient :

$$N_{TOT} = N_a \cdot \frac{H_k}{H_M}$$

Le pourcentage de succès h (*Hit Ratio*) d'accès aux k objets les plus populaires est donné par :

$$h = \frac{N_{TOT}}{N_a} = \frac{H_k}{H_M}$$

En fixant la valeur de h, la valeur de k peut être choisie telle que :

$$\ln k = h \cdot \ln M \quad \Rightarrow \quad k = M^h \tag{3.17}$$

Exemple : Pour M =1000 objets entreposés sur le MSS, pour atteindre un pourcentage de succès de **80%** auprès d'un QR, le MSS doit charger dans les QR ses **251 premiers** objets.

Phase III : Diffusion des QR

Cette phase est la plus importante. Le MSS, au lieu d'évaluer la fonction du coût d'accès:

$$\sum_{i \in N \setminus H} \sum_{j \in H} c_{ij} x_{ij} \quad \text{sujet à } \alpha \leq \sum_{i \in N \setminus H} x_{ij} \leq \beta \quad (j \in H)$$
$$\text{et} \sum_{j \in H} x_{ij} = 1 \qquad (i \in N \setminus H) \tag{3.18}$$

effectue plutôt une diffusion de la liste des QR qu'il vient juste de désigner et d'approvisionner. Nous adoptons plutôt cette stratégie de diffusion pour :

- éviter au serveur de devoir assigner à chaque UM non désignée, un QR pour le service de données ;
- réduire le temps de calcul quant à l'achèvement de la procédure de désignation des QR ;
- permettre à chaque UM d'effectuer un choix préférentiel dépendamment du profil d'un QR.

Dans la liste de diffusion se trouve une description du profil de chaque QR de même que celui du MSS qui est également un QR. Le contenu de cette liste comporte les champs indiqués à la Figure 3.4:

Adresse du QR	Coordonnées (X, Y, Z)	Vitesse	Nombre d'objets stockés

Figure 3.4 Profil d'un QR

Chaque UM, dès réception de cette liste, crée une table dans laquelle sont enregistrées les composantes de la liste diffusée. L'UM peut s'associer à un QR selon des critères tels que :

- la proximité du QR : l'UM calcule pour chaque QR, la distance les séparant ;
- la vitesse du QR : l'UM peut décider de choisir un QR ayant une faible vitesse relative par rapport à la sienne ;
- le nombre d'objets stockés : l'UM peut décider d'acheminer ses requêtes à un QR stockant plus d'objets, parce qu'elle estime avoir plus de chance d'obtenir une réponse.

Le QR répondant à l'un ou l'autre des critères de choix de l'UM est celui choisi par cette dernière. Toutes les requêtes de l'UM seront ainsi acheminées vers celui-ci. Nous allons cependant privilégier le premier critère car il est celui qui garantit le délai d'accès le plus court à la ressource requise. Lorsqu'une requête de l'UM ne trouve pas sa réponse auprès de son antémémoire, elle achemine cette requête dépendamment du rang de l'objet requis, au QR le plus proche. Autrement, l'UM achemine la requête au serveur, en prenant soin d'inclure dans cette dernière, son profil actualisé (coordonnées, vitesse). Le MSS peut donc effectuer à intervalle de temps une diffusion de la liste des QR avec leur profil actualisé.

Nombre optimal de QR

La désignation des QR nécessite la connaissance du nombre optimal de QR comme décrit par la contrainte (3.10) de la fonction objectif F_{QR}. Nous allons, dans cette section, déterminer ce nombre optimal de QR tout en lui définissant une borne inférieure et supérieure. Soit N le nombre de requérants à l'époque de désignation t, et k le nombre de QR. En moyenne, N/k UM peuvent accéder à un objet sur un QR, y compris ce dernier. La moyenne de l'espace de stockage total dans le réseau est :

$$\overline{S_{TOT}} = k \cdot \overline{S_{jQR}} + \frac{N}{k} \overline{S_{jUM}}$$

où $\overline{S_{jQR}}$ est la moyenne de l'espace de stockage des QR et $\overline{S_{jUM}}$ celle des UM. En admettant que les UM disposent du même espace de stockage, c'est-à-dire $S_{jQR} = S_{jUM} = S$, on obtient la fonction de stockage :

$$f(k) = \left(k + \frac{N}{k}\right) \cdot S \qquad \textbf{(3.19)}$$

On peut trouver le nombre de QR minimisant la fonction de stockage en faisant :

$$\frac{\partial f(k)}{\partial k} = 0 \quad \Rightarrow \quad k_{\min} = \sqrt{N} \qquad \textbf{(3.20)}$$

Le nombre optimal de QR k_{opt} peut être choisi tel que :

$$\sqrt{N} \le k_{opt} \le N \qquad \textbf{(3.21)}$$

3.4 Mise en antémémoire

Les UM qui ne sont pas désignées QR entretiennent une antémémoire dont la taille est identique à celle des QR. Lorsqu'une UM effectue une requête, elle vérifie s'il est possible d'obtenir cet objet auprès de son antémémoire. En cas d'échec, la requête est acheminée au QR le plus proche ou au serveur de données, dépendamment du rang de l'objet requis. À la réception de la réponse, l'objet est stocké en antémémoire avant d'être servi au requérant, suivant la politique de renouvellement. A cet effet, les données stockées font l'objet de deux politiques de renouvellement : LRU (*Least Recently Used*) et MFU (*Most Frequently Used*). La Figure 3.5 donne une esquisse du format d'un objet stocké en antémémoire.

ObjetID	Objet	Horodate
(a) objet stocké avec LRU		

ObjetID	Objet	Fréquence
(a) objet stocké avec MFU		

Figure 3.5 Format d'un objet stocké en antémémoire

Le champ *ObjetID* représente l'identifiant de l'objet et est identique à celui de l'objet stocké dans la base de données du MSS. Le champ *Horodate* identifie l'époque à laquelle l'objet a été stocké ou accédé dernièrement. Il est mis à jour dans l'un ou l'autre de ces cas, suivant qu'il existe déjà ou qu'il est nouveau. Cette horodate est nécessaire pour la mise en œuvre du mécanisme de renouvellement LRU. Le champ *Objet*, quant à lui, représente l'objet proprement dit. Lorsqu'un objet se trouve déjà en antémémoire, tout accès futur à ce dernier occasionne la mise à jour du champ Horodate. Ainsi, lorsque l'espace antémémoire est rempli, toute autre réponse y sera stockée en remplaçant le plus vieux des objets existants en antémémoire.

Dans le cas du mécanisme MFU, seuls sont stockés les objets dont la fréquence d'accès est la plus élevée. Ainsi, lorsque l'espace alloué à l'antémémoire est rempli d'objets, une nouvelle arrivée d'objets occasionne le remplacement de celui en antémémoire ayant la plus faible fréquence d'accès.

Quant aux QR, lorsqu'ils effectuent une requête, l'existence de l'objet requis est vérifiée dans l'espace de stockage dont ils disposent. En cas d'échec, la requête est acheminée au serveur de données. Ainsi, contrairement à une UM, un QR n'entretient pas une antémémoire. Les objets de la base de données du serveur y sont stockés de façon statique. Cependant, un QR est muni d'un mécanisme de compression de données de façon à stocker plus d'objets qu'une UM. Dépendamment de l'algorithme de compression de données utilisé, un QR pourra stocker deux à trois fois plus d'objets qu'une UM.

3.5 Maintenance des QR

La mise en place des QR nécessite aussi une procédure de maintenance relative à leur relocalisation. En effet, la perte d'un QR est une faiblesse dont il faudra tenir compte. Ainsi, dû à leur rôle de pseudo serveur de données, les QR sont appelés à consommer plus d'énergie qu'une UM ordinaire. Ils peuvent donc disparaître dans le temps, de sorte qu'une procédure de relocalisation est nécessaire pour étendre le nombre de QR, dépendamment de la topologie du réseau. Cette procédure de désignation de QR

doit donc être évaluée périodiquement.

Une expansion du nombre de QR est effectuée dépendamment du nombre de QR encore actifs. L'intervalle de temps entre deux évaluations doit être assez large pour permettre au MSS de faire un apprentissage de la topologie du réseau, à travers les multiples requêtes des usagers.

3.5.1 Période de localisation

L'exécution de la procédure de localisation des QR est une opération nécessitant la prise en compte d'informations telles que :

1. la localisation de l'UM (coordonnées géographiques) ainsi que sa vitesse qui sont des paramètres dynamiques, car variant avec le changement de topologie du réseau ; de ce fait, le MSS entretient une table dans laquelle elle crée ou maintient à jour une entrée pour chaque UM lorsque cette dernière effectue une requête;
2. le nombre de requérants qui est un paramètre important pour la détermination du nombre de QR à installer; il est déterminé par la taille de la table des UM entretenue par le MSS. À chaque requête d'une UM ayant déjà une entrée dans la table, une mise à jour de son profil est effectuée.

À quelles conditions s'effectue la procédure de désignation des QR ?

Le MSS évalue à intervalle de temps préfixé, la dynamique de la topologie du réseau, et déclenche la procédure de désignation dans chacun des cas suivants:

1. le nombre de requérants de l'intervalle de temps Δt est supérieur à celui ayant servi lors de la précédente désignation durant Δt-*1*. Soient $N_{\Delta t}$ et $N_{\Delta t-1}$ les nombres de requérants à chacun de ces intervalles de temps. Sachant que la désignation de QR devrait réduire le nombre de requêtes parvenant au MSS en distribuant la charge du service de données sur les QR désignés, une désignation est déclenchée quand :

$$N_{\Delta t} > N_{\Delta t - 1} \qquad \textbf{(3.22)}$$

2. le nombre de QR encore actifs à Δt ne vérifie plus :

$$k \geq \sqrt{N} \qquad \textbf{(3.23)}$$

Remarque

Notons qu'à l'issue de chaque procédure de désignation, le MSS réinitialise la table contenant le profil des requérants pour un usage futur, tout en maintenant une liste des UM qu'il a auparavant désignée QR. Une esquisse du contenu de cette table est donnée au Tableau 3.1.

Tableau 3.1 Profil des UM requérantes

Host	Adresse	CoordX	CoordY	Vitesse	Énergie	Stockage	Densité
UM1	192.128.0.1	800.00	200.00	1m/s	50mWhr	200KB	6
UM2	192.128.0.2	500.00	150.00	5m/s	30mWhr	100KB	2
⋮	⋮	⋮					

Lors d'une procédure de relocalisation, une UM active désignée précédemment QR garde son statut. Une relocalisation équivaudrait juste à une extension ou non du nombre de QR précédemment désignés et encore actifs. En effet, un QR peut disparaître pour défaut d'énergie ou s'isoler du reste du réseau pour toutes autres raisons. Pour maintenir une certaine marge quant au nombre de QR, nous allons choisir un optimum à :

$$k_{opt} = 1.5\sqrt{N} \qquad \textbf{(3.24)}$$

Le système (QR+ UM+MSS) est un système asservi au nombre de QR, perturbé par la défaillance des QR (manque d'énergie, indisponibilité, imprévisibilité etc.). La réponse du système est représentée par le nombre de QR actifs dans le réseau en tout temps. La Figure 3.6 représente le diagramme fonctionnel du système asservi.

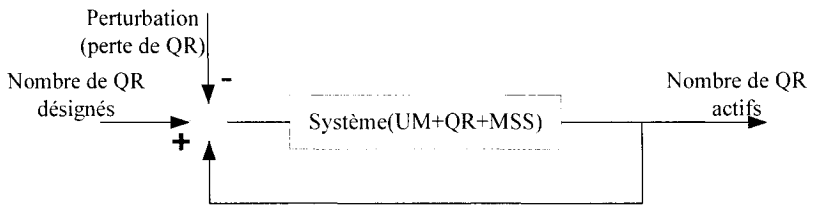


Figure 3.6 Diagramme fonctionnel du système asservi au nombre de QR

Une mise à jour du profil des QR est nécessaire pour maintenir la consistance du positionnement géographique de ces derniers. Pour s'assurer de la présence des QR désignés, le MSS maintient une table qu'il met à jour lors des requêtes acheminées par ces derniers.

Chaque QR désigné, en marge des requêtes qu'il effectue, signale à intervalle de temps sa présence au serveur de données en envoyant un message du type *HELLO*. Ceci permet au serveur, en cas d'indisponibilité d'un QR, de détruire son profil dans la table des QR de manière à assurer une cohérence, quant au nombre de QR encore actifs dans le réseau. Ainsi, le serveur peut, à des intervalles de temps prédéterminés diffuser la liste des QR encore actifs dans sa table, aux UM qui, dès réception maintiennent également leur table de QR à jour.

De la même façon, lorsqu'un QR quitte un MSS pour un autre, il s'annonce à ce dernier à l'aide du même message HELLO. Ainsi, le serveur peut l'enregistrer dans sa table de QR et le publier dans sa liste de diffusion.

Remarque

Un avantage de la réplication partielle est de permettre aux UM temporairement déconnectées pour économiser de l'énergie, de toujours accéder à des données valides après une reconnexion. Un QR tenant lieu de pseudo serveur de données connaîtrait rarement des périodes d'inactivité; la probabilité qu'un QR se déconnecte pour économiser de l'énergie est donc très faible.

3.6 Compression de données

Nous avons montré plus haut que le pourcentage de succès d'accès à l'information désirée augmentait avec la taille de l'espace de stockage. Ainsi, pour augmenter ce ratio, la mise en place d'un mécanisme de compression efficace est nécessaire. Lors de la procédure de chargement des QR, à cause du volume important d'objets à transmettre et à stocker, nous utiliserons un algorithme de compression de données existant tel que celui de *Lempel-Ziv-Welsch* (*LZW*). Notre choix s'est porté sur ce dernier car c'est une méthode de compression indépendante de la nature de la source à compresser contrairement à celui de *Huffman* qui doit d'abord faire une étude statistique de l'occurrence des caractères avant codage. Pendant que la méthode de Huffman compresse un texte avec un ratio de 20%, LZW compresse image et texte avec un ratio de 40 à 60% [26]. Par conséquent, si nos objets sont des mélanges de textes et d'images (comme c'est le cas pour les pages web), la compression LZW serait mieux adaptée. Elle est d'ailleurs très populaire et déjà utilisée dans les utilitaires de compression de données tels que *Gzip* et *Winzip*. Nous rappelons de manière brève le principe de la compression LZW.

La méthode LZW consiste à remplacer par quelques bits, un mot, une phrase ou même un paragraphe entier. Ces bits sont constitués de manière unique à l'aide d'un dictionnaire créé au fur et à mesure que s'effectue le codage. On commence avec un dictionnaire rempli avec tous les caractères existants (0 à 255). Les nouveaux mots ajoutés au dictionnaire commencent donc au numéro 256. Les pseudo-codes des procédures de compression et décompression sont esquissés à la Figure 3.7. Les notations utilisées sont les suivantes : $\oplus$ est l'opérateur de concaténation; D représente le dictionnaire des symboles rencontrés; s, t, u, a, b sont des variables.

Dans le processus de chargement des QR, au lieu de compresser les objets les uns après les autres, nous ferons plutôt une compression groupée des objets de manière à réduire le temps de chargement. Une fois reçus par le QR, ces objets sont maintenus compressés pour permettre le gain en espace de stockage.

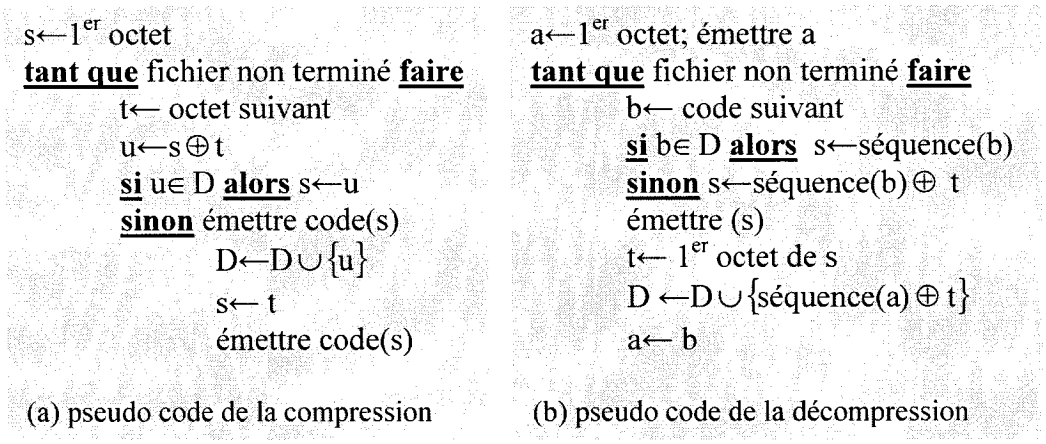


Figure 3.7 Algorithme de compression et de décompression LZW

3.7 Algorithmes

Dans cette section, nous faisons un résumé de la mise en œuvre du modèle, sous forme d'algorithmes. Toutes les fonctionnalités relatives au MSS sont mises en œuvre au sein d'un agent servant d'intermédiaire entre ce dernier et les UM. Quant aux UM, elles communiquent également avec le MSS par l'intermédiaire d'un agent mandaté pour jouer soit le rôle de QR, ou le simple rôle d'UM.

3.7.1 Agent du MSS

C'est l'entité de gestion des interactions avec le réseau de QR et d'UM. L'agent du MSS dispose de trois temporisateurs : l'un dénommé *TQR* pour évaluer la procédure de localisation des QR, un autre dénommé *TBR* qui, à expiration permet la publication du profil des QR et le dernier, *TOUT*, qui à expiration permet au serveur de s'assurer de la disponibilité des QR précédemment désignés. La Figure 3.8 illustre l'algorithme régissant les fonctionnalités du MSS.

Par ailleurs, le MSS maintient en permanence dans le temps deux tables : une table pour mettre à jour le profil des QR, qu'il diffuse à l'expiration du temporisateur *TBR* ou à l'issue de la procédure de désignation des QR; une table pour mettre à jour le profil des UM lorsque ces dernières effectuent des requêtes. Cette table est purgée à

l'issue de chaque procédure de désignation de QR. L'agent du MSS incorpore aussi un engin pour la gestion des interactions avec la base de données.

3.7.2 Agent de l'UM

Son rôle varie dépendamment du fait que l'UM est élue QR ou non. Dans son rôle de QR, il gère l'espace de stockage réservé pour charger les données les plus populaires du MSS, ainsi qu'un engin de compression et de décompression des données chargées.

Dans son rôle d'UM toute simple, il gère l'espace de stockage réservé pour l'antémémoire, et se contente juste d'acheminer ses requêtes au QR le plus proche se trouvant dans sa table *(QRTable),* ou au serveur. Cette table contient le profil des QR reçus de la diffusion faite par l'agent du MSS. La Figure 3.9 donne un aperçu de l'algorithme régissant les fonctionnalités de l'agent du côté de l'UM.

Pour évaluer la fiabilité du modèle que nous proposons, nous allons aborder également les points suivants :

- mise en place d'un réseau ad hoc dans lequel aucune UM ne dispose d'antémémoire ;
- mise en place d'un réseau ad hoc dans lequel toutes les UM disposent d'une antémémoire ;
- mise en place d'un réseau ad hoc où le schéma Greedy-S [17] décrit dans le chapitre précédent est appliqué ;
- mise en place d'un réseau ad hoc dans lequel notre modèle sera appliqué.

Cette évaluation se fera en termes de comparaison des différents schémas précités.

(A) Quand une requête arrive

Si l'objet *ObId* est requis par un QR

Mettre à jour la table *QRTable*

Servir l'objet requis au QR

Sinon

Si l'adresse de l'UM existe dans la table *UmTable*

Mettre à jour la table *UmTable*

Sinon créer une entrée dans *UmTable*

Servir l'objet requis à l'UM

Fin Si

Fin Si

(B) À l'expiration de TQR

N(t) = TailleDe(*UmTable*)

Si $N(t) > N(t-1)$ ou $k_{min} < 1.5\sqrt{N(t)}$

Pour $k := 1,\ldots N(t)$ **Faire**

Évaluer la fonction objectif F_k

Retourner $\min\{F_k\}$

Désigner les QR

Pour $j := 1,\ldots |H|$ **Faire**

Charger les QR

Publier la liste des profils des QR désignés

Vider la table *UmTable*

Mettre à jour la table *QRTable*

Fin Si

(C) À l'expiration de TBR

Publier la liste des profils des QR de la table *QRTable*

(D) À l'expiration de TOUT

Détruire le profil des QR devenus indisponibles, et diffuser la nouvelle liste

Figure 3.8 Algorithme exécuté par l'agent du MSS

(A) UM élue QR

a. Lorsqu'une requête arrive
   Si *Objet* se trouve dans QRStore
      Servir *Objet* au requérant
   Sinon
      Effectuer la requête de *Objet* au MSS
   Fin Si
b. **Tant que** le QR reçoit un avis de chargement **Faire**
      Décompresser *Objet* et récupérer *ObjetID*
      Mettre *Objet* dans *QRStore*
c. Lorsque le QR reçoit une réponse
      Servir la réponse à l'application requérante

(B) L'UM n'est pas élue QR

a. Lorsqu'une requête arrive
      Récupérer l'ID de l'objet requis
      Si ID < NbreObjet
            Retourner l'adresse du QR le plus proche dans *QRTable*
            Acheminer la requête de *Objet* à cette adresse
      Sinon
            Acheminer la requête au MSS
      Fin Si
b. Lorsque le QR reçoit le profil des QR élus
      Si *QRtable* est vide
            Créer une entrée pour chaque QR élu
      Sinon
            Mettre à jour le profil de chaque QR dans *QRtable*
      Fin Si
c. Lorsque l'UM reçoit une réponse
      Si *CacheStore* n'est pas plein
      Si *Objet* existe dans *CacheStore*
      Mettre à jour *Objet*
      Sinon
            Créer une entrée pour *Objet* dans *CacheStore*
      Fin Si
      Servir la réponse à l'application requérante
   Sinon
      Si *Objet* existe dans *CacheStore*
            Mettre à jour Objet
      Sinon
            Supprimer l'objet dont l'horodate est le plus ancien
            Créer une entrée pour *Objet* dans *CacheStore*
      Fin Si
      Servir la réponse à l'application requérante
   Fin Si

Figure 3.9 Algorithme exécuté par l'agent de l'UM

CHAPITRE IV

IMPLÉMENTATION ET RÉSULTATS

Dans ce chapitre, nous allons procéder à l'évaluation de performance du modèle proposé au chapitre précédent. Dans un premier temps, nous présenterons l'environnement d'implémentation et de simulation ainsi que les détails des structures de données utilisées par les algorithmes du modèle proposé. Une fois cet exposé fait, nous procèderons à la mise en place d'un plan d'expériences en définissant au passage les métriques que nous utilisons pour l'évaluation des performances du modèle. Enfin, outre la présentation des résultats de simulations, nous effectuerons une analyse de ces derniers et situerons les performances du modèle par rapport aux performances d'autres solutions proposées dans la littérature.

4.1 Environnement d'implémentation et de simulation

Cette section donne une description de l'environnement de développement de notre modèle ainsi que les outils utilisés.

4.1.1 Environnement de développement

La plate-forme utilisée pour la mise en œuvre de notre solution est du type *PC*. C'est une machine munie d'un processeur *Intel Pentium* de 1.80 GHz et d'une mémoire vive de 256 Mo, avec un système d'exploitation *Windows XP*. L'éditeur *Visual C++ 6.0* est utilisé pour la programmation et la compilation s'effectue avec l'engin *nmake* faisant partie de la suite *Visual Studio .NET (2003) de Microsoft*. La mise en place d'une base de données a nécessité l'usage du système de gestion de base de données *Microsoft Access*.

4.1.2 Le simulateur

Le simulateur utilisé est celui de Qualnet, conçu en particulier pour les réseaux sans fil. Il est développé à l'*Université de Californie à Los Angeles (UCLA)* et se base sur un langage de simulation parallèle dénommé PARSEC (*PARallel Simulation Environment for Complex system*) à évènements discrets très similaire au langage C. Qualnet se présente sous la forme d'une architecture en couches, similaire à celle de TCP/IP. La communication s'effectue uniquement entre les couches adjacentes à l'aide d'API prédéfinies. Puisque Qualnet est un simulateur à évènements discrets, la mise en place d'un protocole s'effectue à l'aide d'une machine d'états dont les transitions s'effectuent à l'occurrence d'évènements; un évènement est défini comme un incident causant le changement d'état ou l'activation d'une action précise. Les cinq couches du simulateur sont les couches Application, Transport, Réseau, Liaison (MAC) et Physique. La couche Application est responsable de générer du trafic. Elle est particulièrement intéressante pour l'implémentation de notre modèle. La couche Transport garantit un service de transmission de bout-en-bout aux protocoles de la couche Application. Les protocoles TCP, UDP et RSVP-TE y sont implémentés pour les besoins de la couche Application. La couche Réseau, quant à elle, implémente les protocoles de routage tels que AODV (*Ad hoc On Demand Vector*) et DSR (*Dynamic Source Routing*) spécifiquement conçus pour les réseaux ad hoc, ainsi que d'autres protocoles sur lesquels nous ne nous attarderons pas ici. La couche MAC implémente plusieurs protocoles de liaisons tels qu'IEEE 802.3, IEEE 802.11 et le CSMA. La couche Physique assure un accès brut au canal de transmission et implémente un grand nombre de modèles de propagation radio et de mobilité.

Le modèle en couches du simulateur Qualnet permet d'intégrer facilement de nouveaux protocoles. Chaque protocole exécute sa machine d'états dépendamment de la couche à laquelle il est implémenté [27]. La Figure 4.1 présente une vue d'ensemble de la structure d'une couche du simulateur.

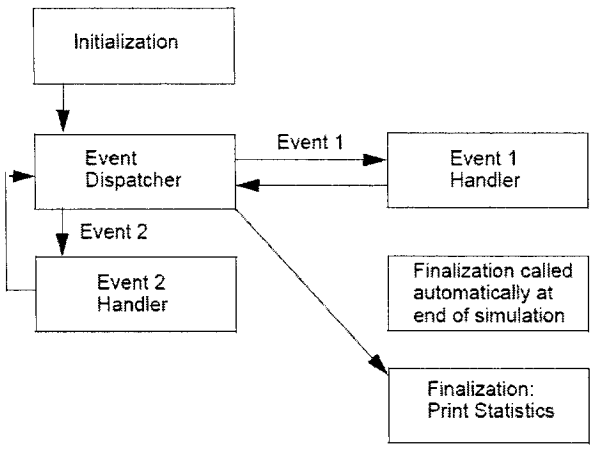


Figure 4.1 Structure d'une couche du simulateur Qualnet

Chaque couche est implémentée comme un gestionnaire d'évènement qui reçoit des évènements sous forme de structure de données appelées messages contenant le type d'évènement ainsi que les informations associées. Le gestionnaire d'évènements détermine alors le type d'évènement puis en crée ou non un nouveau.

À chaque couche, un protocole est initialisé par une fonction qui reçoit en entrée les données relatives à son fonctionnement et effectue sa configuration. Ainsi, quand un évènement survient, le gestionnaire d'évènements en détermine le type et effectue l'aiguillage vers le protocole concerné. À la fin de chaque simulation, une fonction de finalisation est appelée pour chaque protocole et pour chaque nœud pour l'impression des statistiques. Un évènement de fin de simulation est généré automatiquement pour permettre la transition vers l'état final des protocoles impliqués dans la simulation. En résumé, chaque protocole doit implémenter les trois fonctions suivantes : une fonction d'initialisation, un gestionnaire d'événements, et une fonction de finalisation.

4.2 Méthodologie d'implémentation

Dans cette section, nous allons décrire les détails de l'implémentation du modèle proposé au sein du simulateur. Nous commencerons par justifier le choix de la couche d'implantation du modèle, puis nous décrirons les structures de données utilisées.

4.2.1 Choix de la couche d'implémentation

Un type d'application comme celle de la mise en antémémoire est généralement implanté au sein de la couche application du modèle OSI. Ceci est déjà le cas dans les réseaux filaires, et se justifie par le fait que ce type d'application est relié directement au trafic des usagers, notamment d'Internet. Dans cette même lignée, nous allons implanter notre modèle dans la couche applicative du simulateur, sous forme d'agent mandaté pour la gestion d'antémémoire au niveau de chaque UM. Serveur et client seront représentés chacun par un agent pour l'exécution des tâches afférentes à la mise en antémémoire.

4.2.2 Agent du serveur

La Figure 4.2 montre la structure interne de l'agent du serveur.

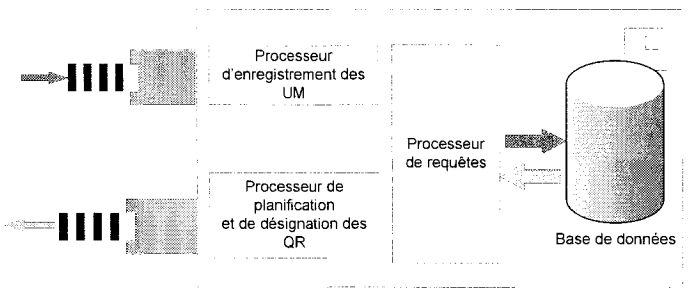
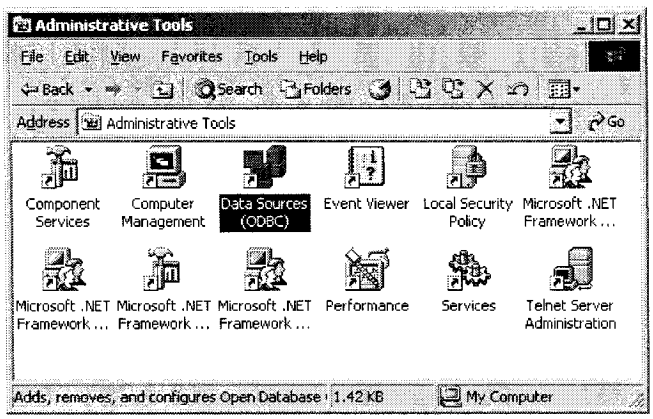
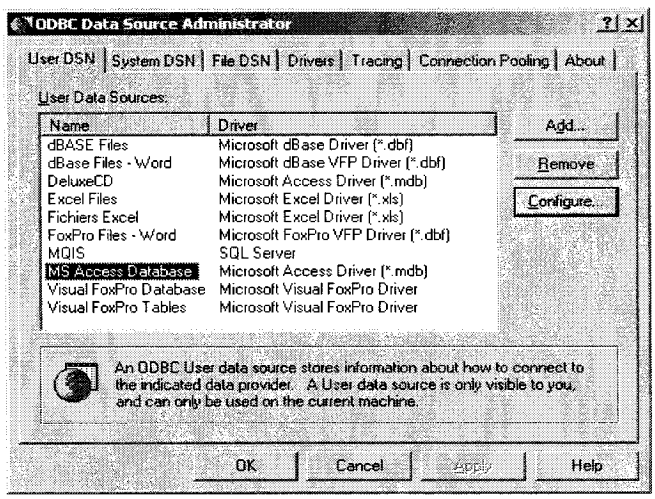


Figure 4.2 Structure interne du serveur de données (MSS)

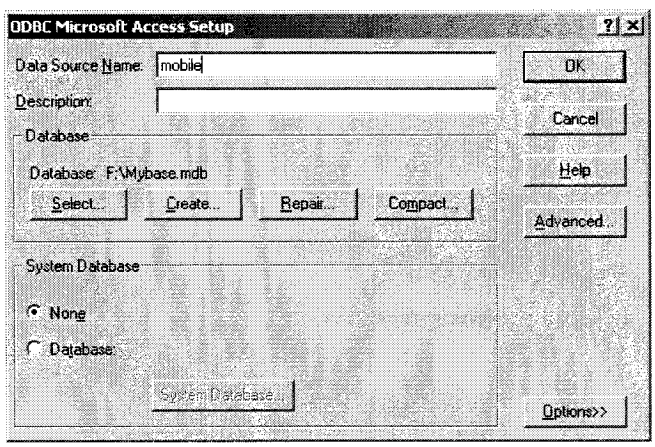
Il se compose d'une Base de données *Microsoft Access* : une liaison ODBC (*Open DataBase Connectivity*) permet la connexion avec la base de données par l'intermédiaire de la fonction *SrvagProcessQuery(Node* node, char* request, SrvagData* dataPtr)* représentant le processeur de requêtes. Cette fonction sert l'objet requis en l'acheminant vers une queue de sortie dénommée *outQueue*. La Figure 4.3 donne un aperçu de l'intégration de la base de données au simulateur.



(a) Ajout d'une connexion ODBC



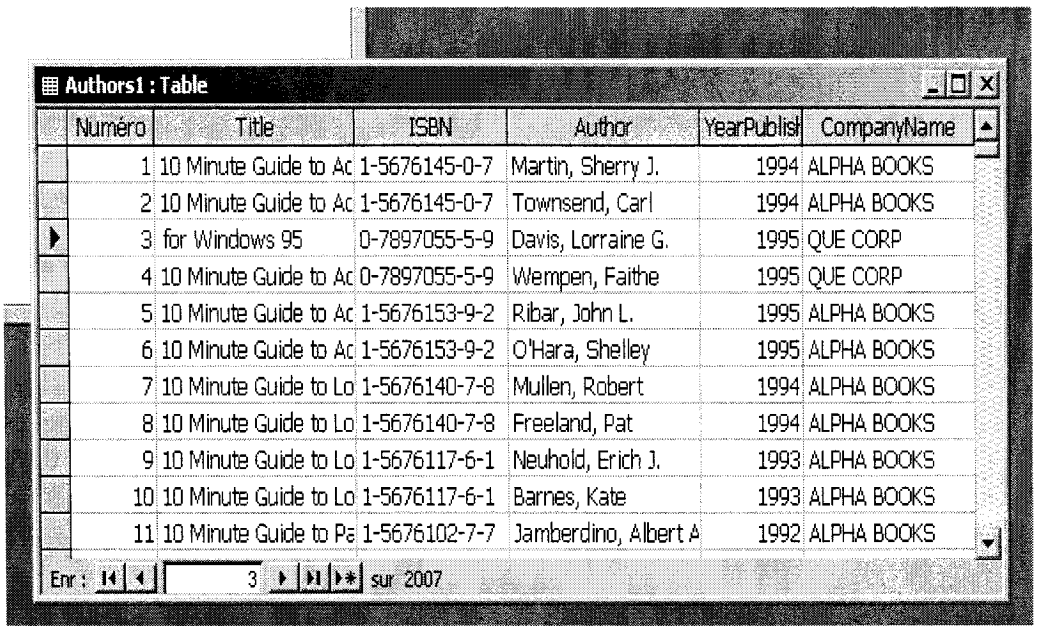
(b) Sélection du driver



(c) Sélection de la base de données.

Figure 4.3 Intégration de la base de données au simulateur

Cette base de données contient une table dénommée "Authors1" comprenant cinq champs, et répertoriant un total de 1000 objets de taille moyenne égale à 1 Ko. Un bref aperçu du contenu de cette table est donné à la Figure 4.4.



Authors1 : Table

Numéro	Title	ISBN	Author	YearPublish	CompanyName
1	10 Minute Guide to Ac	1-5676145-0-7	Martin, Sherry J.	1994	ALPHA BOOKS
2	10 Minute Guide to Ac	1-5676145-0-7	Townsend, Carl	1994	ALPHA BOOKS
3	for Windows 95	0-7897055-5-9	Davis, Lorraine G.	1995	QUE CORP
4	10 Minute Guide to Ac	0-7897055-5-9	Wempen, Faithe	1995	QUE CORP
5	10 Minute Guide to Ac	1-5676153-9-2	Ribar, John L.	1995	ALPHA BOOKS
6	10 Minute Guide to Ac	1-5676153-9-2	O'Hara, Shelley	1995	ALPHA BOOKS
7	10 Minute Guide to Lo	1-5676140-7-8	Mullen, Robert	1994	ALPHA BOOKS
8	10 Minute Guide to Lo	1-5676140-7-8	Freeland, Pat	1994	ALPHA BOOKS
9	10 Minute Guide to Lo	1-5676117-6-1	Neuhold, Erich J.	1993	ALPHA BOOKS
10	10 Minute Guide to Lo	1-5676117-6-1	Barnes, Kate	1993	ALPHA BOOKS
11	10 Minute Guide to Pa	1-5676102-7-7	Jamberdino, Albert A	1992	ALPHA BOOKS

Enr : 3 sur 2007

Figure 4.4 Contenu de la table de la base de données

L'agent du serveur intègre aussi une queue d'entrée dénommée *InQueue* qui reçoit les requêtes et les achemine à leur arrivée pour être traitées par le processeur de requêtes. Une fois la requête traitée par le processeur, la réponse est acheminée vers la queue de sortie. Cette dernière dénommée *outQueue* achemine les réponses aux requêtes en les préparant à la diffusion. Elle est parcourue à l'expiration d'un temporisateur. Le serveur dispose aussi d'un temporisateur qui, à expiration de la période d'apprentissage de la configuration du réseau, lui permet désigner les UM devant héberger un réplica de la base de données. En effet, la période d'apprentissage représente l'intervalle au cours duquel l'agent comptabilise le nombre d'UM requérantes. Durant cette période, l'agent crée une structure de données décrivant le profil de chaque requérant. Un engin de compression et de décompression de données muni de l'utilitaire *GZIP* est utilisé pour

procéder au chargement des UM désignées QR. Il offre une efficacité de compression de 1/3.

Quant à la mise à jour du profil des requérants, une table dénommée *UmTable* est utilisée. La librairie STL de C++ offre une souplesse quant à la manipulation des tables par usage de conteneurs et d'itérateurs du type *vector*. Dans cette table sont enregistrées les informations relatives à une UM, telles que ses coordonnées géographiques, sa vitesse, la taille de sa mémoire ainsi que le nombre de voisins dont elle dispose. Pour la mise à jour du profil des UM élues QR, une table dénommée *QrTable* est utilisée. Cette table est servie dans la réponse à une requête de l'UM. De cette façon, chaque UM dispose en tout temps du profil de chaque QR élu. La structure de données manipulée par l'agent du serveur est illustrée à la Figure 4.5.

Cette structure de données possède des attributs qui définissent la configuration de l'agent du serveur, à savoir son état *state* (émission, réception de données, libre), son adresse *NodeAddress*, et le port d'émission/réception (*ConnectionId, protocol*). Quant aux attributs de position, de vitesse et de densité, ils sont indiqués par les variables *density, speed, xsource, ysource*. Les attributs de type *vector* représentent les tables manipulées par l'agent du serveur, pour le service des requêtes. Il faut noter que chaque requête est identifiée par le triplet (*ReqId, Req, ReqAddress*) représentant respectivement le numéro de l'objet requis, le libellé de la requête ainsi que l'adresse de l'UM requérante. Le reste des attributs représente ceux permettant d'effectuer la désignation des QR à l'expiration du temporisateur *TQR*.

(a) Structure de données du serveur

```
typedef struct struct_srvag_str
{
int state;
int connectionId;
short sourcePort;
int   protocol;
NodeAddress MSSAddr;
unsigned short seed[3];

int density;
double speed;
double xSource;
double ySource;

char req[MAX_STRING_LENGTH];
int ReqID;
NodeAddress ReqAddress;
int CurrentUmTableSize;
int PreviousUmTableSize;
int CurrentQrTableSize;
float nextime;

vector <string> UmTable;
vector <string> QrTable;
vector <string> QrPreTable;
queue <Message*> inQueue;
queue <string> outQueue;
queue <string> ReqQueue;
BOOL QRReqStart;

int QROpt;
int OptNow;
float speedF;
float cacheF;
float densityF;
string QrList;
} SrvagData;
```

(b) Structure de données de l'UM

```
typedef struct struct_umag_str
{
int state;
short sourcePort;
int protocol;
int connectionId;

NodeAddress clientAddr;
NodeAddress MSSAddr;
clocktype sessionStart;
clocktype sessionFinish;
BOOL sessionIsClosed;
clocktype meanInterval;
clocktype startTime;
clocktype endTime;

double xSource;
double ySource;
double speed;

BOOL IsQR;
BOOL QRSet;
BOOL stat;
BOOL ReqStart;

char type;
unsigned short seed[3];
char req[MAX_RESP_BUFF];
char TimeStamp[15];

int density;
int ReqID;
int ObjId;
int ServerHit;
int QRHit;
int CacheHit ;
int NumOfReq;
int NumOfResp;
int CacheSize;

vector <NodeAddress> neighborTable;
vector <string> QrTable;
vector <string> QRStore;
queue <string> QRqueue;
vector <Cache*>CacheStore;

} UmagData;
```

(c) Structure de données d'antémémoires

```
typedef struct
{
char TimeStamp[12];
int ObjId;
char Objet [MAX_RESP_BUFF];

} Cache;
```

Figure 4.5 Structures de données

Procédure de désignation des QR

Il existe une période d'apprentissage au bout de laquelle le serveur procède au choix des QR. À cet effet, le serveur détermine le nombre optimal de QR, à partir du

nombre d'UM préalablement enregistrées dans la table *UmTable*, ceci à l'expiration de la période d'apprentissage. Le format du profil de chaque UM enregistrée dans cette table est représenté à la Figure 4.6.

Figure 4.6 Structure de données du profil d'une UM enregistrée par le serveur

Par la suite, toute réponse à une éventuelle requête se voit dotée aussi de la liste des QR que le serveur vient juste de désigner. Il faut noter que les réponses aux requêtes sont compressées à l'aide de l'engin GZIP avant d'être acheminées vers la queue de sortie. Dès réception de la réponse, l'UM effectue une décompression de la réponse du serveur, récupère la réponse utile puis effectue une mise à jour de sa table de QR dont le format est illustré à la Figure 4.7.

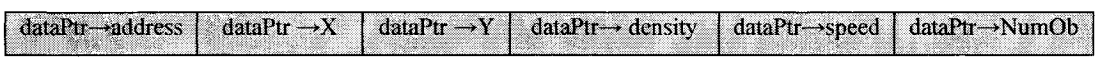


Figure 4.7 Profil d'un QR enregistré par le serveur ou l'UM

De cette façon, l'UM obtient à chacune des réponses qu'elle reçoit du serveur une situation quasi-précise de l'emplacement des QR élus. Elle peut donc acheminer dès lors ses requêtes au QR le plus proche en calculant la distance les séparant. Il faut mentionner qu'avant d'envoyer sa requête, que ce soit au serveur ou à un QR, l'UM effectue une vérification de l'existence de ce dernier dans son antémémoire, ceci à l'aide de la fonction *CheckObjectInCache(Node* node, UmagData* dataPtr)*.

Suivant le nombre d'objets chargés dans ledit QR et l'ID de l'objet requis, l'UM achemine sa requête vers ce dernier. Le QR peut ainsi servir l'UM requérante depuis son entrepôt. Quant au QR, chaque fois qu'il effectue une requête, une vérification de l'existence de l'objet est effectuée dans son entrepôt *QRStore*, ceci à l'aide de la fonction *CheckQRStore(Node* node,UmagData* dataPtr)*. Cette fonction retourne l'objet si ce dernier se trouve dans l'entrepôt; sinon la requête est acheminée au serveur.

4.2.3 Agent de l'UM

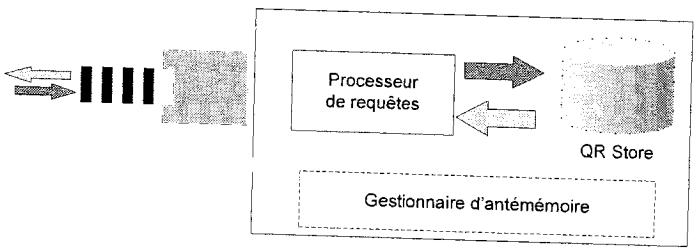


Figure 4.8 Structure interne de l'agent d'une UM

L'agent de l'UM, quant à lui s'articule autour de trois fonctions majeures, comme décrites par la structure interne présentée à la Figure 4.8.

Gestionnaire d'antémémoire

Cette fonction est dédiée à l'antémémoire et au stockage des objets servis par le serveur. Elle implémente les mécanismes de remplacement LRU et MFU. Pour cela, chaque objet est mémorisé à l'aide de trois champs tels qu'indiqués par la structure de données de la Figure 4.5. Le champ *TimeStamps* indique la dernière date à laquelle l'objet *Objet* de numéro *ObjId* a été sollicité. Un conteneur de type *vector* est utilisé pour mémoriser ces objets de type *Cache**. Les règles régissant la mise en antémémoire d'un objet sont résumées par l'esquisse de code présentée à la Figure 4.9.

```
static void MFU(Node* node,UmagData* dataPtr)
{
      if(dataPtr->CacheStore.size() < dataPtr->CacheSize)
      {
            CacheNewObject(node,dataPtr);
      }
      else
      {
          if(dataPtr->Frequency > min{dataPtr->Frequency})
          {
              RemoveMinObject(node,dataPtr);
              CacheNewObject(node,dataPtr);
          }
      }
}
```

(a) Politique de renouvellement MFU

```
static void LRU(Node* node,UmagData* dataPtr)
{
      int id = CheckObjectInCache(node,dataPtr);

      if(dataPtr->CacheStore.size() < dataPtr->CacheSize)
      {
            if (id)
            {
                  UpdateObject(node,dataPtr);
            }
            else
            {
                  CacheNewObject(node,dataPtr);
            }
      }
      else
      {
            if(id)
            {
                  UpdateObject(node,dataPtr);
            }
            else
            {
                  RemoveOldObject(node,dataPtr);
                  CacheNewObject(node,dataPtr);
            }
      }
```

(b) Politique de renouvellement LRU

Figure 4.9 Esquisse de code des politiques de renouvellement de l'UM

Distribution de Zipf

Cette fonction implémente la distribution de Zipf et définit ainsi le motif d'accès aux objets requis par l'UM. En effet, dans un fichier texte *'requete.txt'*, nous avons formulé, pour tous les champs d'un objet, une requête SQL de la forme :

```
Select OID, Title, ISBN, Author, YearPublished,
CompanyName from Authors where OID =…
```

Chaque UM, pour initier une requête, génère un nombre entier à l'aide de la distribution de *Zipf* dans l'intervalle **[1,1000]**. Une fois généré, ce nombre correspond au numéro de la requête que l'UM lira dans le fichier de requêtes.

Le motif d'émission de la requête suit une distribution de poisson dont une implémentation est déjà disponible dans le simulateur. L'UM émet la requête en appelant la fonction *enterUmagTimerExpired(...)* qui contrôle l'émission des requêtes de façon périodique, la période d'émission correspondant au temps entre deux requêtes.

Réception de réponses

Une fonction dénommée *enterUmagReceiveResponse()* est appelée à chaque fois que l'UM reçoit une réponse du serveur. Elle implémente comme le serveur, un engin de compression et de décompression de données utilisant l'utilitaire Gzip. Lorsque l'UM est élue QR, elle entretient un fichier dans lequel sont stockés les objets chargés par le serveur, et ceci sous format compressé. À taille de mémoire équivalente, une UM élue QR stocke trois fois plus d'objets qu'une UM simple, à cause du mécanisme de compression de données.

4.3 Plan d'expériences et de simulation

Dans cette section, nous effectuons la mise en œuvre d'un plan d'expériences pour évaluer les performances du modèle proposé que nous dénommons *QRScheme*. En marge de la configuration des scénarios et des différents paramètres de la simulation, le plan d'expériences définit un certain nombre d'indices de performance que nous allons mesurer en fonction de facteurs clés décrits plus loin. Il faut mentionner qu'un seul facteur sera varié à la fois.

4.3.1 Définition des indices de performance

Pour évaluer la performance du modèle, nous allons le comparer à une configuration sans mise en antémémoire (*No Caching*), à celle d'une mise en antémémoire systématique (*Caching*) (où toutes les unités disposent d'une mémoire dédiée pour le stockage d'objets fréquemment accédés), et enfin avec le schéma *Greedy-*

S. Les indices de performance que nous avons retenus pour évaluer la performance du modèle sont :

- le *délai moyen d'accès* : il représente le temps écoulé entre l'émission d'une requête par une UM et la réponse à cette requête;
- le *débit efficace* : si q_{tot} désigne le nombre total de requêtes effectuées par une UM et q_{succ} le nombre de réponses obtenues avec succès, l'expression du débit est donnée par :

$$\mathrm{T} = \frac{q_{succ}}{q_{tot}} \cdot 100\% \qquad \textbf{(4.1)}$$

- le *Local Hit Ratio* : il représente le pourcentage de requêtes ayant trouvé une réponse dans l'espace antémémoire d'une UM, comparativement au total des réponses obtenues. Soit q_{cache} le nombre de requêtes ayant trouvé une réponse en antémémoire, l'expression du *Local Hit Ratio* (*LHR*) est donnée par :

$$LHR = \frac{q_{cache}}{q_{tot}} \cdot 100\% \qquad \textbf{(4.2)}$$

- le *Server Hit Ratio (SHR)* : il représente l'ensemble des requêtes ayant trouvé leurs réponses auprès du serveur de données.

Nous unifierons la dénomination de ces deux derniers indices sous celle plus simple de *Hit Ratio* (*HR*).

4.3.2 Choix des facteurs

Les résultats des mesures effectuées avec les différents indices ci-dessus définis seront donnés en fonction de six facteurs dont nous étudierons les effets sur le fonctionnement du modèle. Ce sont :

- la densité ou encore le nombre de nœuds composant le réseau : ce facteur permettra de déterminer le comportement du modèle lorsque le nombre de nœuds dans le réseau augmente. Les niveaux de ce facteur seront choisis entre un minimum de 20 et un maximum de 80, ceci par pas de 10 nœuds ;

- la vitesse des nœuds pour délimiter les performances du modèle : le choix de ce facteur tient au fait que la forte mobilité des nœuds crée une instabilité des liens. Ce sera ainsi une manière d'évaluer la réaction du modèle aux perturbations causées par cette instabilité. À cet effet, les vitesses MIN_SPEED et MAX_SPEED seront prises dans l'ensemble V = {[0, 5], [5, 10], [10, 15], [15, 20], [20, 25]} ;
- la taille allouée pour l'antémémoire d'une UM : ce facteur est choisi pour évaluer l'impact de la taille de l'espace antémémoire et de la réplication sur les indices prédéfinis. En particulier, cette taille est exprimée comme une fraction du nombre d'objets stockés dans la base de données. Elle sera choisie dans l'ensemble T = {2%, 3%, 4%, 5%, 6%, 7%, 8%, 9%, 10%} ;
- la taille de la base de données choisie dans l'ensemble {300, 400, 500, 600, 700, 800, 900, 1000} ;
- le facteur de Zipf (θ) : puisque la distribution de Zipf est celle utilisée pour le motif d'accès aux objets entreposés sur la base de données, il est nécessaire d'en varier le coefficient θ pour en évaluer l'influence sur le comportement du modèle; il sera pris dans l'intervalle [0,1] et ceci par pas de 0.2 ;
- le protocole de routage pour évaluer le comportement du modèle vis-à-vis du protocole de routage : pour cela, nous allons tester le modèle avec un protocole réactif tel que AODV (*Ad hoc On Demand Distance Vector*) et un autre proactif, nommément le protocole OLSR (*Optimized Link State Routing Protocol*).

4.3.3 Configuration de la simulation

Dans le cadre de notre série de simulations, nous considérons un réseau mobile ad hoc distribué uniformément sur une surface de 600m × 600m. Les unités mobiles (UM) utilisent une interface de communication sans fil de type 802.11b en mode ad hoc avec un débit d'interface de 2 Mbps. Le protocole de couche MAC ne requiert pas l'utilisation des messages RTS et CTS pour accéder au médium de communication. De plus, l'usage de la même technique d'accès au médium permet d'avoir une même portée de transmission, soit 50 mètres. Par ailleurs, la propagation des signaux se fait suivant un

modèle en espace libre. Le serveur est représenté par un nœud fixe dont le débit d'interface est de 11 Mbps avec une portée de transmission couvrant l'aire de simulation. Le modèle de mobilité choisi pour les simulations est le *Random-Waypoint* déjà largement utilisé dans la littérature, car approchant plus la réalité. En effet, dans le modèle *Random-Waypoint*, une unité mobile choisit une destination à l'intérieur de la surface définie. Le déplacement de l'unité mobile en direction de sa destination se fait à une vitesse constante choisie selon une distribution uniforme située entre MIN_SPEED et MAX_SPEED. Une fois que l'unité atteint sa destination, elle attend sur place pour un temps déterminé par PAUSE_TIME avant de choisir une nouvelle destination et de s'y rendre. Deux variables permettent de contrôler le mouvement des unités mobiles : la vitesse v et la durée de la pause p. Les mesures sont prises lorsque l'antémémoire de l'UM est remplie et que la politique de remplacement est en vigueur. En outre, dans le cas du modèle QR, les mesures sont effectuées après la désignation des QR et le remplissage des antémémoires des UM non désignées QR.

Tableau 4.1 : Paramètres de simulation

Paramètres	Valeurs par défaut	Intervalle
Taille de la Base de données	1000 objets	300 -1000
Nombre de noeuds	60	20-80
Taille moyenne par objet	1k	
Distribution des nœuds	Uniforme	
Temps inter requêtes	10s	
Distribution de Zipf θ	0.95	0-1
Taille des Réplica / Cache	2% de la base de données	1-10%
Modèle de mobilité	Random-Waypoint	
Aire de simulation (m)	600 x 600	
Débit (Mbps)	2	
Portée de transmission (m)	50	
Intervalle de vitesse (m/s)	0-5	0-5, 5-10,10-15,15-20, 20-25
Temps de simulation (min)	20	
Protocole de routage	AODV	AODV, OLSR

Dans le but d'approcher un intervalle de confiance de 95% dans les mesures, nous effectuerons une série de 7 simulations pour chaque indice de performance. Le

Tableau 4.1 résume la configuration des principaux paramètres utilisés durant les simulations. Les facteurs qui varient d'une simulation à l'autre sont surlignés en gris.

4.3.4 Scénarios de simulation

Dans cette section, nous présentons deux scénarios clés qui font l'objet de nos simulations.

Scénario 1 :

C'est le scénario de base. Un agent serveur est monté sur un seul nœud, dont la vitesse est variable dans l'intervalle [0,2] tandis que celle des autres nœuds est maintenue variable dans l'intervalle [0, 5]. Le serveur est disposé dans un coin de l'aire de simulation. Les autres nœuds mobiles, sont équipés d'un agent UM et leur nombre est maintenu variable. La Figure 4.10 décrit cette configuration. Les UM non QR sont représentées en blanc, celles élues QR sont représentées en bleu, et le serveur en rouge.

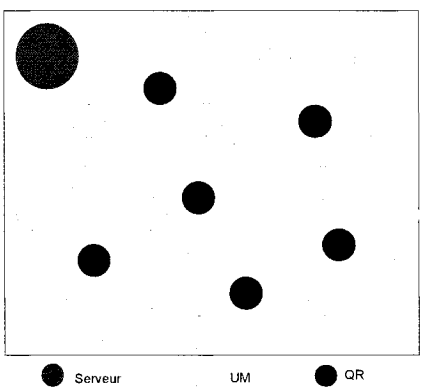


Figure 4.10 Configuration du scénario de base

Dans le cas d'une mise en antémémoire simple (Caching) et du schéma Greedy-S, les UM peuvent recevoir des réponses de leur antémémoire. Le délai de réponse est ainsi considéré nul. Dans celui avec des QR actifs, les UM reçoivent des réponses soit de leur antémémoire, soit depuis le QR le plus proche ou encore à partir du serveur. Lorsque le QR effectue lui-même une requête qui trouve sa réponse dans son espace de stockage, le délai de réponse est considéré également nul.

Scénario 2 :

Comme le scénario précédent, la vitesse des nœuds est maintenue variable dans l'intervalle [0, 10], tandis que celle du serveur est maintenue variable dans les intervalles de l'ensemble R = {[0, 2], [2, 4], [4, 6], [6, 8], [8, 10]}. L'objectif de ce scénario est d'observer le comportement du modèle lorsque la vitesse du serveur augmente par rapport à celle des autres nœuds du réseau.

4.4 Analyse des résultats de simulation

Les simulations visant à évaluer les performances du modèle proposé sont, à cette étape, achevées. Dans cette section, nous présentons les résultats obtenus en termes de comparaison entre les différentes configurations implémentées Un grand nombre de résultats étant généré, nous présentons ici ceux qui sont d'intérêt pour comprendre le comportement du modèle.

4.4.1 Scénario 1

Il s'agit du scénario de base. C'est le scénario le plus important. La vitesse des nœuds est comprise entre 0 et 10 m/s, avec un temps de pause de 30 secondes. La vitesse du serveur, est choisie entre 0 et 2 m/s avec le même temps de pause que les autres nœuds.

Délai moyen d'accès aux objets

Cet indice de performance est un indicateur de performance clé que nous observons en fonction de tous les facteurs que nous avons définis plus haut. Ainsi, la Figure 4.11 montre le délai moyen en fonction de la densité de nœuds dans le réseau.

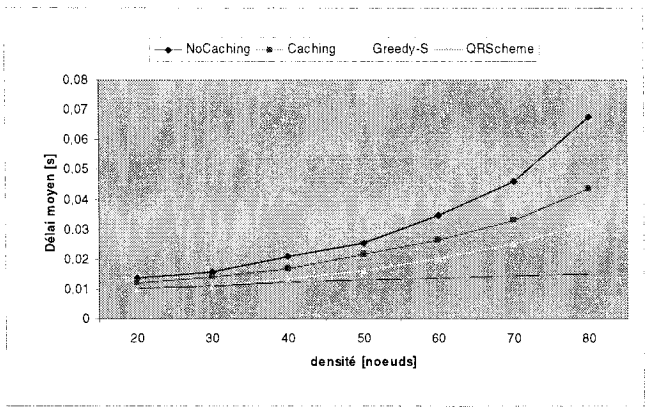


Figure 4.11 Délai moyen de réponse en fonction de la densité

On note, à partir des résultats illustrés à la Figure 4.11, que le délai d'accès aux objets entreposés dans la base de données augmente avec la densité de nœuds dans le réseau. L'allure de ces courbes presque régulière peut être approchée par une fonction de la forme :

$$\text{Délai} = \rho \cdot e^{\alpha \cdot \text{density}} \ [\text{seconde}] \qquad \textbf{(4.3)}$$

La réplication partielle (*QRScheme*) offre cependant des délais plus faibles, comparativement aux autres schémas. En particulier, en effectuant une régression exponentielle d'ordre 2 sur ces courbes, on obtient les résultats de la Figure 4.12 avec les équations de la régression afférentes à chaque schéma. Le modèle proposé *QRScheme* est donc plus performant comparativement aux autres configurations, à cause de son coefficient α plus faible que ceux des autres. Avec la régression effectuée, il est possible à partir de ces résultats, d'effectuer une liaison entre la configuration de mise en antémémoire systématique et celle de la réplication partielle, telle que :

$$\Delta\alpha = f\left(\sqrt{\text{density}}\right) \qquad \textbf{(4.4)}$$

avec $\Delta\alpha$ représentant la différence entre les coefficients de mise en antémémoire systématique et de réplication partielle.

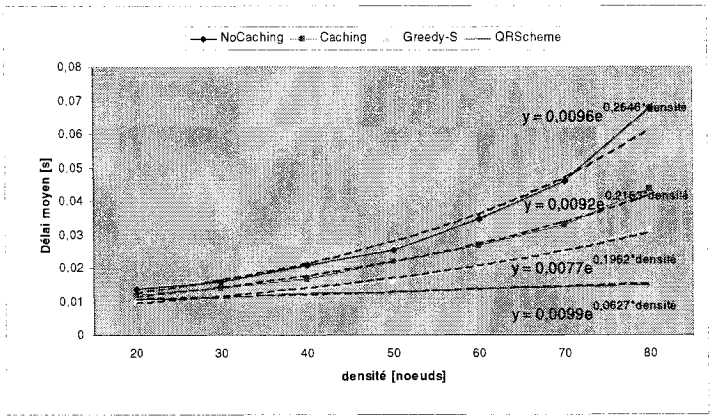


Figure 4.12 Effet de la régression sur la variation de densité

La Figure 4.13 montre la variation du délai moyen en fonction de la taille allouée à l'antémémoire d'une UM, *TailleCache*. Cette taille est exprimée en pourcentage de celle de la base de données.

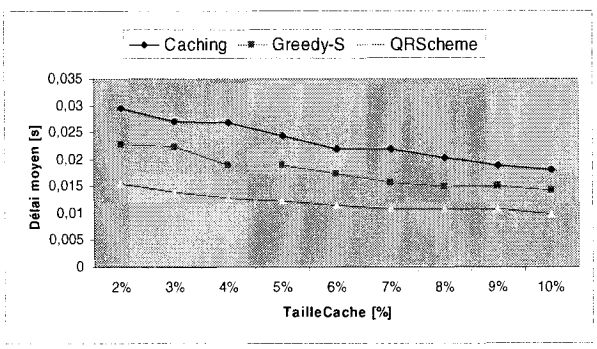


Figure 4.13 Variation du délai moyen en fonction de la taille de l'antémémoire

Ainsi, on note qu'avec l'augmentation de cette taille, le délai moyen d'accès aux objets diminue et ceci de façon régulière. Cette situation est due au fait qu'il y a plus d'objets stockés en mémoire. La stratégie *QRScheme* offre cependant des délais plus faibles à cause d'objets obtenus par les nœuds auprès de QR qui leur sont plus proches. En particulier, il est possible d'approcher cette tendance à l'aide d'une régression exponentielle d'ordre 2 telle qu'illustrée par la Figure 4.14.

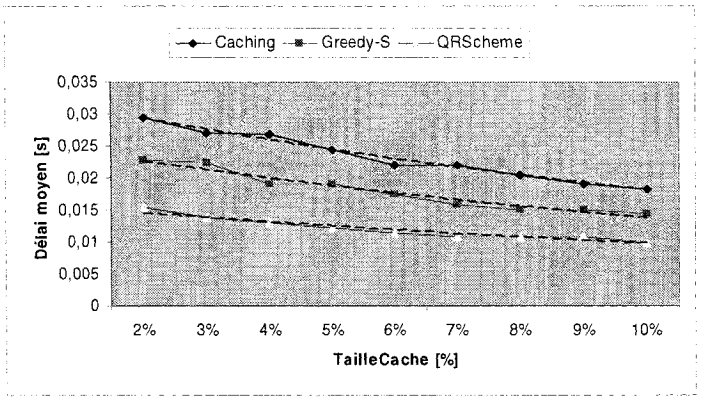


Figure 4.14 Effet de la régression sur la variation de la taille d'antémémoire

Le délai moyen d'accès aux objets diminue lorsque la taille allouée aux antémémoires des UM croît. La taille de l'antémémoire a donc un effet significatif sur le délai d'accès aux objets. La réplication partielle, elle, suit aussi cette même tendance.

La Figure 4.15 montre, quant à elle, la variation du délai moyen en fonction de la taille de la base de données. En effet, on observe que ce délai croît de manière assez lente lorsque la taille de la base de données augmente. La réplication partielle (QRScheme) produit des délais moyens plus faibles.

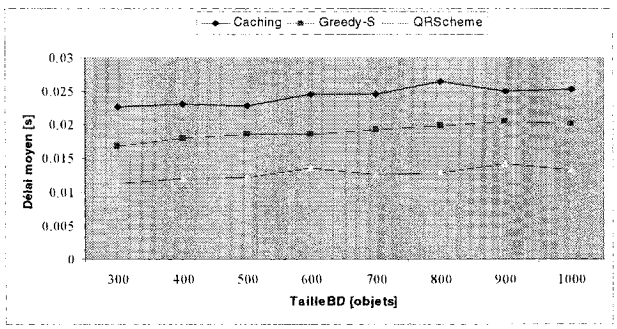


Figure 4.15 Variation du délai moyen en fonction de la taille de la base de données

La tendance de cette augmentation quasi régulière illustrée à la Figure 4.16 est telle qu'il est possible à l'aide d'une régression linéaire d'ordre 2, de l'approcher par la fonction :

$$\text{Délai} = \alpha \cdot (\text{BDSize}) + \beta \quad [\text{seconde}] \qquad \textbf{(4.5)}$$

Avec l'effet de cette régression illustrée à la Figure 4.16, la performance de QRScheme par rapport aux autres schémas, peut être expliquée par la pente plus faible de sa droite de régression, et ceci lorsque la taille de la base de données augmente.

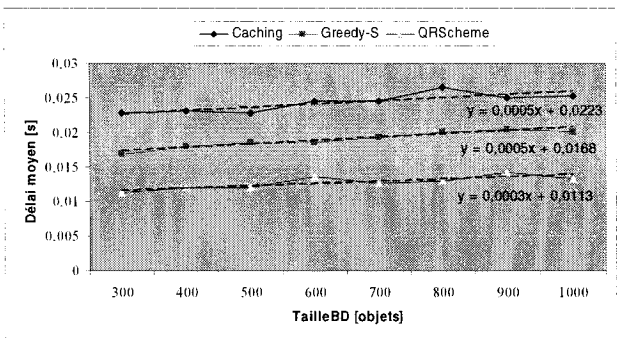


Figure 4.16 Effet de la régression sur la variation de la taille de la base de données

La Figure 4.17 quant à elle, illustre le délai moyen en fonction de la vitesse des nœuds. En particulier, les niveaux de vitesse pour les différents nœuds ont été choisis dans six intervalles de vitesse V = {[0, 5], [5, 10], [10, 15], [15, 20], [20, 25], [25, 30. La vitesse du serveur quant à elle est maintenue variable dans l'intervalle [0, 2].

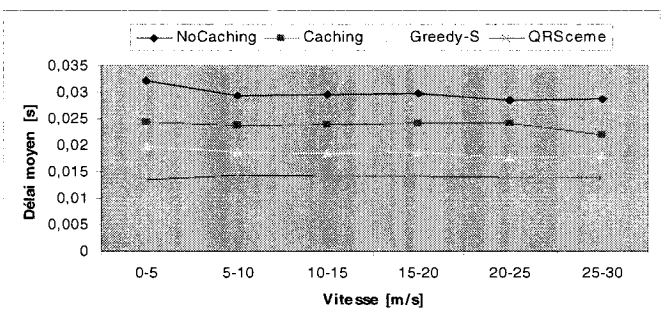


Figure 4.17 Variation du délai moyen en fonction de la vitesse des nœuds

Avec le temps de pause maintenu à 30 secondes, on note de nouveau une performance du délai moyen dans le cas de la réplication partielle (*QRScheme*) par

rapport aux autres configurations. Cette performance peut être expliquée par le fait que dans leur mouvement, les nœuds se retrouvent souvent dans le voisinage d'un QR leur assurant le service de données dans des délais courts. Cependant, on note que l'effet de la variation du délai en fonction de la vitesse des nœuds est quasi-nul.

La mobilité des nœuds n'affecte pas de façon significative le délai moyen d'accès aux objets de la base de données.

La Figure 4.18 montre la variation du délai moyen en fonction du facteur de Zipf. Rappelons que ce facteur définit le motif d'accès des UM aux objets de la base de données et est utilisé pour décrire les habitudes d'accès des usagers d'Internet aux différents sites web. Lorsque ce coefficient croît, les accès des usagers se focalisent sur les objets de rang plus faible. Avec la réplication partielle (*QRScheme*), ce coefficient est mis à 1, de façon à répliquer sur les QR désignés par le serveur de données, les *n* premiers objets de sa base de données. Grâce à l'engin de compression, un QR peut stocker trois fois plus d'objets que l'antémémoire d'une UM ordinaire. On observe que, malgré la variation du facteur de zipf, la réplication partielle garantit toujours des délais plus faibles que ceux des autres schémas. Cependant, pour des valeurs de ce facteur, supérieures à 0.6, les délais sont nettement plus courts que pour les autres schémas.

Le facteur de zipf, lorsqu'il croît, affecte donc le délai d'accès des UM aux objets d'une base de données en ce sens qu'il l'améliore en réduisant sa valeur.

Dans le motif d'accès aux serveurs web le facteur de zipf est mesuré égal à 0.98.

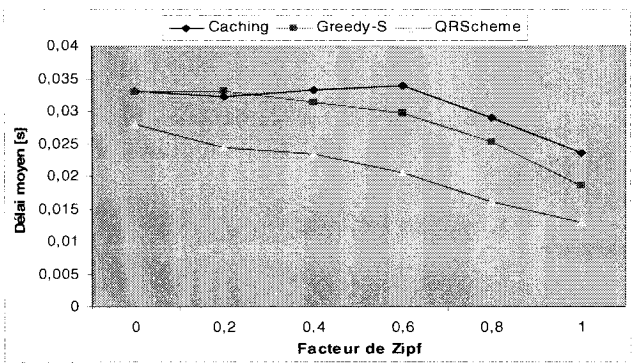


Figure 4.18 Variation du délai moyen en fonction du facteur de zipf

La Figure 4.19 montre la variation du délai moyen en fonction de la densité de nœuds, avec l'usage du protocole de routage proactif OLSR. On note que les performances obtenues avec le protocole AODV utilisé pour générer les résultats précédents, ne sont plus les mêmes. La réplication partielle, bien qu'elle soit meilleure que la mise en antémémoire simple, ne garantit pas des délais plus faibles que le schéma Greedy-S. Cependant, la tendance du délai quant à l'augmentation de la densité est la même que pour le protocole AODV. Cette tendance confirme ainsi une fois de plus que le délai moyen d'accès aux objets de la base de données augmente exponentiellement avec cette densité.

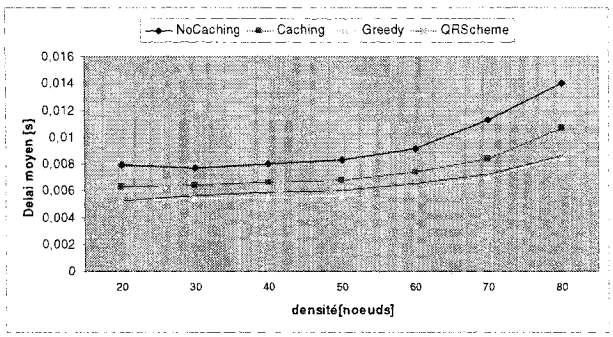


Figure 4.19 Variation du délai en fonction de la densité de nœuds dans le réseau

Débit efficace

Cet indice mesure le nombre de réponses obtenues avec succès aux diverses requêtes effectuées. En effet, la Figure 4.20 montre la variation du débit efficace en fonction de la densité des nœuds du réseau. On note que celui-ci, dans le cas de la réplication partielle (*QRScheme*), est meilleur que le débit offert dans les autres cas et augmente avec la densité de nœuds dans le réseau. Cette amélioration est beaucoup plus visible lorsque cette densité est supérieure à 40 nœuds.

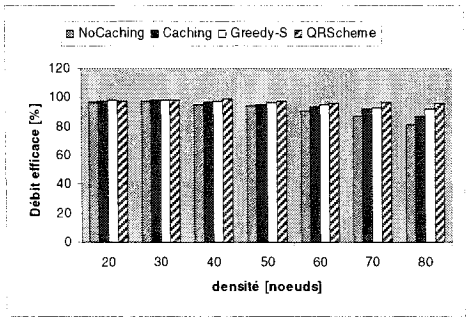


Figure 4.20 Variation du Débit efficace en fonction de la densité

La Figure 4.21 montre la variation du débit efficace en fonction de la vitesse des nœuds. Cette mesure est faite dans le but d'estimer comment le modèle de la réplication partielle réagit aux perturbations causées par la mobilité incontrôlable des nœuds. En effet, le débit efficace connaît aussi une dégradation progressive avec l'augmentation de la vitesse des nœuds comparativement au schéma Greedy-S. Cette baisse est prononcée lorsque la vitesse des nœuds est supérieure à 10 m/s (équivalent de la vitesse d'une voiture). Elle peut s'expliquer par le fait que la forte mobilité des UM occasionne des ruptures fréquentes de liens durant le service de données par les QR. Cependant, le débit efficace reste meilleur comparativement au schéma *Caching*

La vitesse des nœuds a donc un effet perturbateur sur le modèle, en ce sens qu'elle affecte le Débit efficace en la diminuant de façon prononcée et ceci, lorsqu'elle croît.

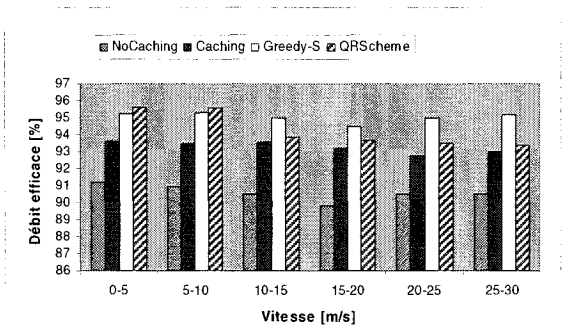


Figure 4.21 Variation du Débit efficace en fonction de la vitesse des nœuds

La Figure 4.22 montre la variation du Débit efficace en fonction du facteur de zipf. On note à partir de ces résultats que la réplication partielle de données (*QRScheme*), améliore le débit efficace par rapport à la mise en antémémoire simple d'au moins 2% en moyenne et ceci, lorsque ce facteur croît. Un maximum de débit est obtenu lorsque ce facteur est égal à 1.

Le Débit efficace est meilleur pour des valeurs élevées du facteur de zipf (typiquement supérieures à 0.6), et plus particulièrement avec une réplication partielle de données.

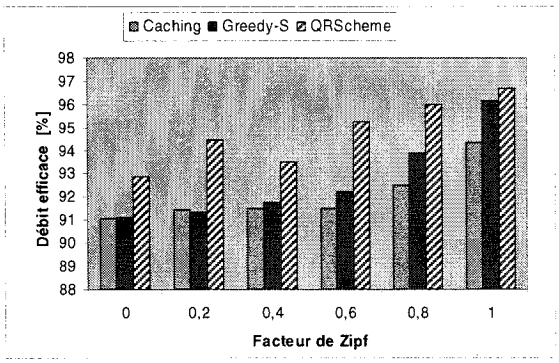


Figure 4.22 Variation du Débit efficace en fonction du facteur de zipf

La Figure 4.23 quant à elle montre la variation du Débit efficace en fonction de la taille de l'antémémoire d'une UM. Elle montre que le débit efficace augmente avec la taille allouée à l'antémémoire. Intuitivement, ce résultat est prévisible car plus d'objets sont stockés en antémémoire au sein de chaque UM. Cependant, la réplication partielle offre de meilleurs résultats du débit efficace, comparativement aux autres schémas. Cette amélioration est en moyenne de 1.5% lorsque la taille de l'antémémoire représente moins de 7% de celle de la base de données et de 0.4% pour des valeurs supérieures ou égales à 7%.

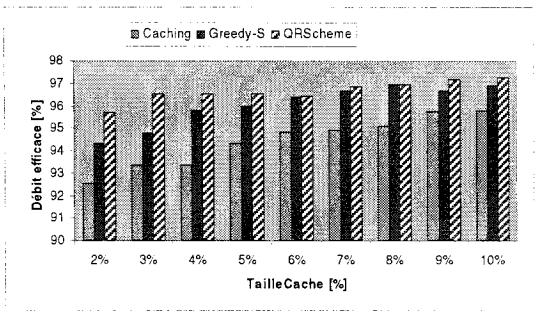


Figure 4.23 Variation du Débit efficace en fonction de la taille de l'antémémoire

La Figure 4.24 montre la variation du débit efficace en fonction de la taille de la base de données. De nouveau, le débit offert par la réplication partielle de données est meilleur, malgré la taille croissante de la base de données. Cependant, il décroît pour tous les schémas avec l'augmentation de la taille de la base de données. Ceci est intuitivement prévisible, car le nombre d'objets populaires est plus important, si bien que l'espace réservé pour les stocker en antémémoire n'est plus suffisant.

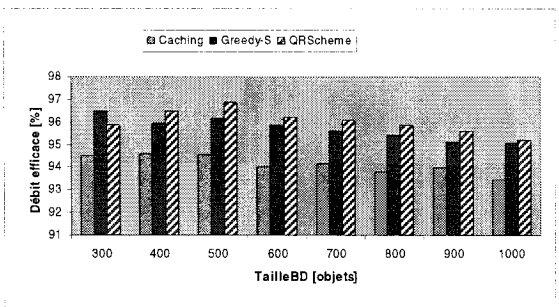


Figure 4.24 Variation du Débit efficace en fonction de la taille de la Base de données

La Figure 4.25 montre la variation du Débit efficace en fonction de la densité de nœuds dans le réseau, et ceci avec le protocole de routage proactif OLSR. On observe ici une dégradation significative du débit efficace lorsque la densité de nœuds dans le réseau augmente. Cette situation peut s'expliquer par le fait qu'il y a un nombre de plus en plus croissant de messages échangés pour que les nœuds apprennent la topologie du

réseau. Donc, cette baisse de performance progressive est imputable au trafic de signalisation du routage.

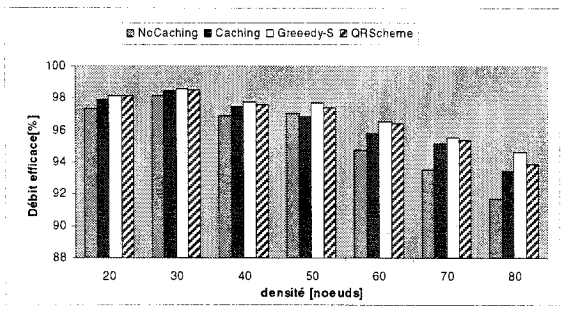


Figure 4.25 Variation du Débit efficace en fonction de la densité (OLSR)

Hit Ratio

Cet indice de performance décrit le pourcentage de réponses obtenues avec succès soit auprès de l'espace antémémoire, soit auprès du serveur de données. La Figure 4.26 montre la variation du Hit Ratio en fonction de la densité de nœuds dans le réseau.

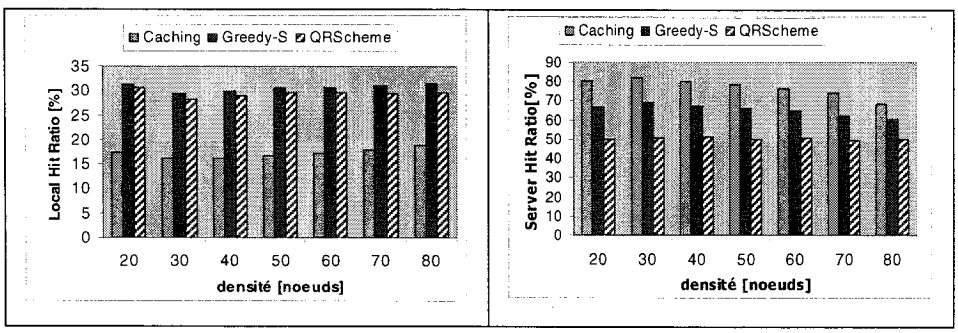


Figure 4.26 Variation du Hit Ratio en fonction de la densité

En effet, on note qu'en moyenne 30% des réponses obtenues avec succès ont été servies depuis l'espace antémémoire comme l'indique le Local Hit Ratio, pour les schémas *QRScheme* et *Greedy-S*. En revanche, ce taux est plus faible dans le cas de la mise en antémémoire systématique où seulement 17% en moyenne des réponses obtenues avec succès sont servies depuis l'antémémoire. Cela peut s'expliquer par le fait que la politique de renouvellement MFU utilisé pour les configurations QRScheme et

Greedy-S, garantit un meilleur accès aux données avec l'usage de la fréquence d'accès à ces dernières.

Par ailleurs, la charge du serveur dans la configuration de la réplication partielle comme le montre le Server Hit Ratio, est beaucoup plus faible comparativement aux autres configurations. En moyenne, cette charge est réduite de 15% par rapport au schéma Greedy-S et de 25% par rapport à la mise en antémémoire systématique Caching.

La réduction de la charge au niveau du serveur est comblée par le service de données assuré aussi par les QR préalablement désignés. La réplication partielle offre donc un avantage certain quant à la réduction du coût d'accès des UM aux objets de la base de données. Une bonne partie des objets (25% en moyenne) est servie depuis les QR.

La réplication partielle de données dans un réseau ad hoc réduit la charge au niveau du serveur de données, mais pourrait causer plus rapidement la défaillance en énergie de l'UM élue QR à cause de son rôle de pseudo-serveur.

La Figure 4.27 montre la variation du Hit Ratio en fonction du facteur de zipf qui l'affecte particulièrement.

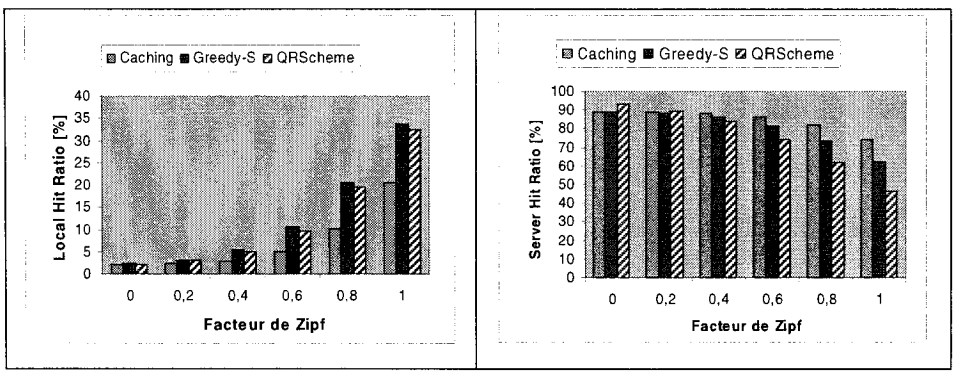


Figure 4.27 Variation du Hit Ratio en fonction du facteur de zipf

En effet, lorsque ce facteur croît, le Local Hit Ratio croît de façon exponentielle, tandis que le Server Hit Ratio connaît l'effet inverse. La charge au niveau du serveur,

décroît plus rapidement avec l'augmentation de ce même facteur dans le cas de la réplication de données que pour les autres configurations.

Lorsque ce facteur est nul, correspondant à un motif d'accès régi par une distribution uniforme, on ne note aucune amélioration vis-à-vis des autres schémas. À l'inverse, lorsqu'il est égal à 1, correspondant à un motif d'accès régi par une distribution comportementale de zipf, non seulement la charge du serveur est réduite de façon drastique, mais encore plus d'objets sont servis depuis les QR et l'espace antémémoire.

Pour les faibles valeurs de ce facteur (entre 0 et 0.6 exclusivement), le serveur est largement sollicité de telle sorte que, même une réplication partielle n'en réduirait pas la charge de façon significative.

La réplication partielle dans les réseaux ad hoc est donc bénéfique seulement pour des valeurs plus élevées du facteur de zipf (0.6 à 1).

La Figure 4.28 montre la variation du Hit Ratio en fonction de la taille allouée à l'antémémoire de l'UM.

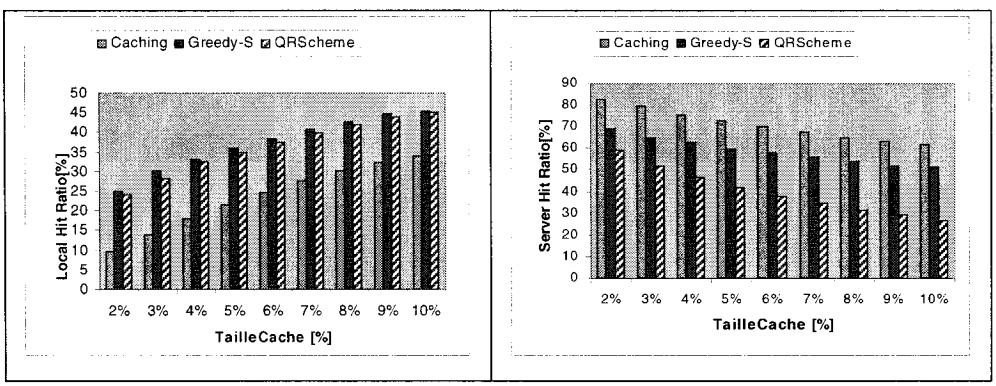


Figure 4.28 Variation du Hit Ratio en fonction de la taille de l'antémémoire

L'effet de l'augmentation de cette taille est significatif. En effet, lorsque la taille de l'antémémoire croît, le pourcentage de succès, que ce soit au niveau de l'antémémoire que des QR, croît de façon régulière. Cela est dû à l'augmentation du nombre d'objets populaires stockés dans cet espace. L'effet sur la charge au niveau du serveur est net, car cette dernière décroît régulièrement avec l'augmentation de l'espace

antémémoire; moins de requêtes sont donc acheminées au serveur. Ces allures régulières peuvent être approchées respectivement pour le LHR et le SHR, par une fonction de la forme :

$$LHR = \lambda \cdot \sqrt{CacheSize} \quad [\%] \qquad \textbf{(4.6)}$$

$$SHR = \alpha \cdot LHR \qquad [\% \qquad \textbf{(4.7)}$$

En outre, la charge du serveur dans le cas de la réplication partielle est réduite grâce au service de données assuré par les QR.

La réplication partielle est encore meilleure lorsque la taille de l'espace antémémoire augmente. Plus d'objets peuvent être répliqués.

La Figure 4.29 montre la variation du Hit Ratio en fonction de la taille de la base de données.

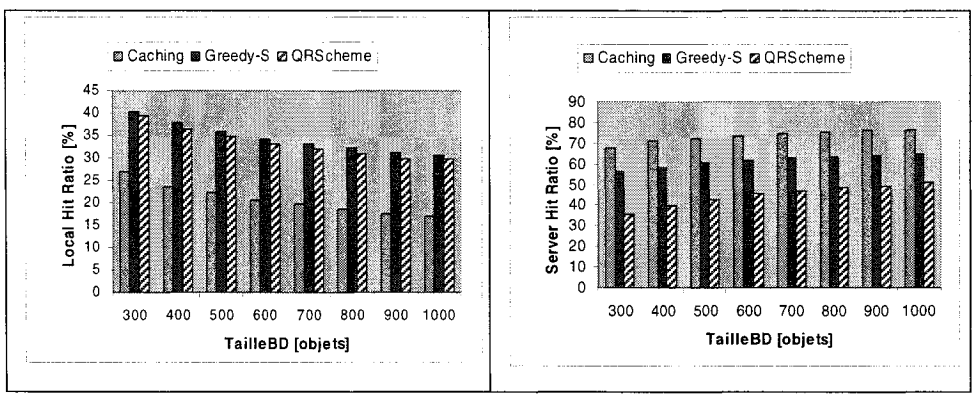


Figure 4.29 Variation du Hit Ratio en fonction de la taille de la base de données

Cette variation est significative sur le Hit Ratio. En effet, pour tous les schémas, on note que lorsque le nombre d'objets de la base de données augmente, de moins en moins d'objets sont servis localement depuis l'espace antémémoire car il y a plus d'objets populaires qui ne peuvent être stockés à cause de la restriction imposée par la taille de cet espace. C'est ce qui explique la diminution exponentielle du Local Hit Ratio. Dans la même logique, puisqu'il y a moins d'objets servis depuis l'antémémoire, la conséquence est que ces objets sont servis soit par le serveur (dans le cas des schémas Caching et Greedy-S) soit par les QR (dans le cas de QRScheme) ; ceci explique

l'augmentation régulière mais très lente du Server Hit Ratio. Cependant, avec la réplication partielle, la charge du serveur est largement moindre que celle des autres schémas.

L'augmentation de la taille de la base de données réduit l'efficacité de la réplication partielle si la taille de l'espace antémémoire n'est pas augmentée. Il faudra donc dimensionner cette taille de manière à réduire la charge du serveur d'un pourcentage bien déterminé.

La Figure 4.30 montre la variation du Hit ratio en fonction de la vitesse des nœuds.

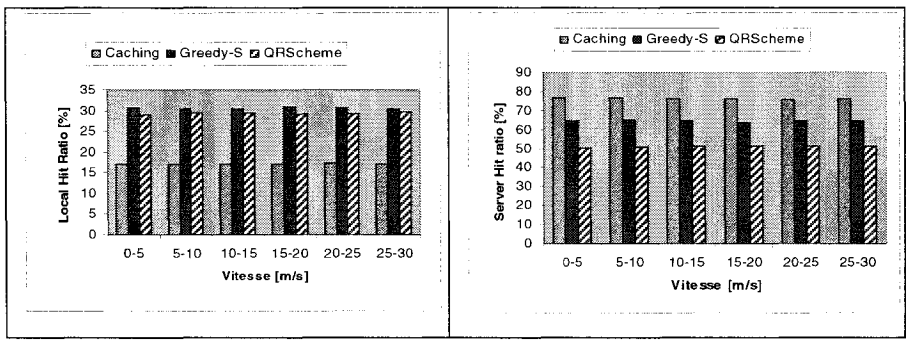


Figure 4.30 Variation du Hit Ratio en fonction de la vitesse des nœuds

Comme pour le délai, on note que, malgré la mobilité incontrôlable des nœuds liée à l'augmentation de leur vitesse, le Hit Ratio ne connaît aucune variation significative. Cette situation s'explique par le fait que chaque nœud n'obtient de réponse que de lui-même ou du serveur (dans le cas des schémas Caching et Greedy-S). En revanche, dans le cas du schéma *QRScheme*, puisque la charge du serveur est constante malgré cette augmentation de la vitesse, cela montre que les UM non QR, dans leur mobilité se sont souvent retrouvées proches de QR. Ceci a garanti cette stabilité de réduction de la charge du serveur de 15% en moyenne par rapport au schéma Greedy-S.

4.4.2 Scénario 2

L'objectif de ce scénario est d'évaluer l'aptitude du modèle à résister à la mobilité incontrôlable du serveur de données, puisque des résultats précédents du débit efficace ont montré une dégradation de ce dernier lorsque la vitesse des UM est

supérieure à 10 m/s. Dans cette optique, nous avons fixé pour les nœuds, une fenêtre de vitesse [0, 10] dans laquelle nous faisons varier une fenêtre de vitesse pour le serveur d'un pas de 2 m/s. La Figure 4.31 montre la variation des indices de performances avec cette nouvelle situation.

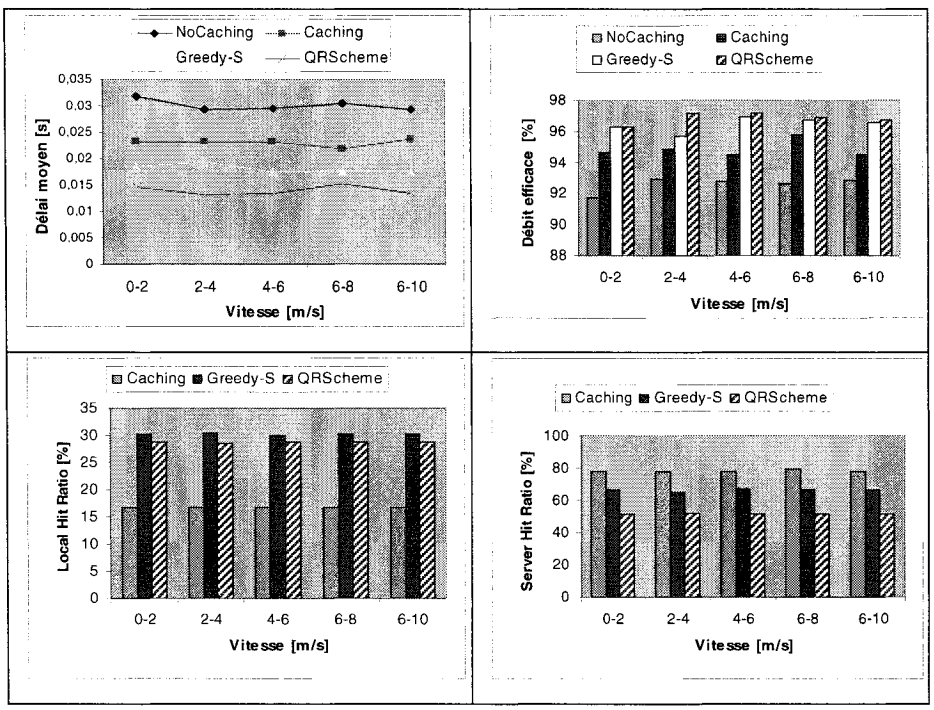


Figure 4.31 Variation des indices de performance en fonction de la vitesse du serveur

Ces résultats montrent une fois de plus que la vitesse du serveur n'a aucune influence notable sur les indices de performance définis.

4.5 Synthèse des performances

Suite à cette phase d'analyse des résultats, une synthèse des résultats obtenus s'impose pour situer la solution proposée. Les résultats satisfont dans l'ensemble aux objectifs préfixés de réduction du coût d'accès aux objets ainsi que l'augmentation du débit efficace, ceci sur l'ensemble des facteurs choisis pour l'évaluation des performances. Ces dernières sont bonnes, comparativement à la mise en antémémoire classique et la stratégie *Greedy-S* proposée dans la littérature. En effet, le délai moyen a

un comportement qu'on peut prédire avec l'augmentation du nombre d'unités mobiles dans le réseau. Il est plus faible dans tous les cas de figure pour la réplication partielle, comparativement aux autres schémas étudiés. D'autre part, la charge du serveur s'est vue réduite de 30% en moyenne par rapport à la mise en antémémoire systématique et de 15% par rapport à la stratégie Greedy-S. Cette charge est distribuée sur l'ensemble des entités hébergeant des réplicas de données du serveur. Quant au Débit efficace, il subit une amélioration de 2% en moyenne sur l'ensemble des cas considérés.

En somme, nous pouvons affirmer que l'évaluation des performances met en lumière la validité de notre approche adoptée pour assurer l'accès aux données dans un réseau ad hoc, à moindre coût.

CHAPITRE V

CONCLUSION

Dans ce mémoire, nous avons traité du problème de la mise en antémémoire de données dans les réseaux ad hoc. Nous avons proposé une stratégie basée sur la réplication partielle de données. Il faut mentionner que l'objectif principal de la mise en antémémoire est de réduire les coûts d'accès aux objets d'une source de données distante et d'améliorer le taux de succès d'accès à l'information désirée. La mise en antémémoire est la stratégie la plus courante, que ce soit dans les réseaux filaires que ceux ad hoc. Par contre, la réplication de données, même si elle est largement utilisée dans les réseaux filaires, sa mise en place dans les réseaux sans fil et plus particulièrement ceux ad hoc, constitue un défi. Dans ce chapitre, nous allons faire une synthèse des travaux réalisés. Nous allons également présenter les limites de la stratégie que nous avons proposée ainsi que quelques pistes pour des travaux futurs dans le domaine.

5.1 Synthèse des travaux

Le problème de la mise en antémémoire dans les réseaux ad hoc consiste à doter les unités mobiles d'un espace de stockage leur permettant de mémoriser les données auxquelles elles accèdent fréquemment. Le premier objectif de cette méthode est de réduire les coûts d'accès à ces données souvent entreposées sur un serveur distant. Le deuxième objectif est d'augmenter le taux de succès d'accès à l'information requise par les unités mobiles. Ces objectifs usuellement fixés pour les réseaux filaires sont, à travers notre recherche, appliqués aux réseaux ad hoc à l'aide d'une stratégie basée sur la réplication partielle de données. Avant d'aller plus loin, il faut mentionner que les travaux de la littérature sur la réplication de données dans les réseaux ad hoc sont rares, sinon presque inexplorés.

Dans ce mémoire, nous avons proposé une combinaison des techniques de mise en antémémoire classique avec la réplication de données pour atteindre les objectifs cités

précédemment. Ainsi, nous avons proposé une architecture dans laquelle sont choisies des unités mobiles disposant d'un profil bien déterminé sur lesquelles seront entreposés des réplicas partiels de données hébergées par une source distante, tandis que les autres se contenteront d'un espace de stockage muni d'une politique de renouvellement MFU (*Most Frequently Used*). Nous avons défini ce profil comme étant un ensemble de paramètres tels que la vitesse de l'UM, son degré de connectivité, l'énergie dont elle dispose, ainsi que son espace de stockage. Nous avons esquissé une formulation mathématique de ce problème de réplication partielle de données en mettant un accent particulier sur les contraintes dont font l'objet les unités mobiles ainsi que le réseau ad hoc qu'elles constituent.

De façon spécifique, nous avons formulé qu'une unité mobile destinée à héberger un réplica partiel de données du serveur distant devra disposer d'une vitesse en dessous de la moyenne pour garantir une stabilité relative quant à son positionnement géographique. Elle doit également disposer d'un degré de connectivité au-dessus de la moyenne, pour mettre les données répliquées dont elle dispose au service d'un grand nombre d'unités mobiles. En outre, elle devra disposer d'une énergie au-dessus de la moyenne pour préserver son indisponibilité prématurée. Par ailleurs, afin d'assurer un stockage consistant de données, des mécanismes de compression sont mis en place pour permettre aux unités mobiles hébergeant des réplicas de stocker trois fois plus d'objets qu'une unité mobile ordinaire, et ceci à espace de stockage égal.

La formulation mathématique du problème effectuée, sa résolution ne peut se faire avec une méthode de résolution exacte. En effet, la mobilité anarchique des entités du réseau ne peut permettre au serveur d'assigner de façon définitive dans le temps une unité mobile à une autre hébergeant des réplicas sans que la topologie du réseau n'ait changé entre temps. D'autre part, le serveur ne peut assigner des objets bien précis de sa base de données à une unité mobile bien précise choisie pour héberger des réplicas de données. Cette difficulté est aussi liée à l'éventualité que la densité d'unités mobiles dans le réseau change et peut augmenter démesurément.

Fort de ces restrictions liées aux comportements intrinsèques des réseaux ad hoc, nous avons adopté une approche coopérative impliquant la collaboration du serveur, des unités mobiles et des unités mobiles hébergeant des réplicas de données. Trois phases sont exécutées pour effectuer la réplication partielle sur des UM clés. Durant la première phase, le serveur lors du service des requêtes d'unités mobiles effectue un apprentissage de la topologie du réseau et choisit les unités mobiles devant héberger des réplicas, suivant un profil d'éligibilité décrit par les paramètres mentionnés plus haut. Durant la deuxième phase, le serveur procède au chargement des unités choisies en données populaires, puis dans la dernière phase, effectue une diffusion de la liste des unités élues ainsi que leurs coordonnées géographiques. Les unités mobiles non choisies évaluent leur proximité avec l'unité mobile la plus proche hébergeant des réplicas de données du serveur. De cette façon, le coût d'accès à un objet est évalué par l'unité requérante avant d'acheminer sa requête.

Pour tenir compte des habitudes des usagers pour l'accès aux données, nous avons utilisé la distribution de zipf qui décrit le comportement d'accès des usagers aux pages des serveurs web. En effet, cette distribution déjà utilisée pour évaluer la popularité des pages web est appliquée au modèle proposé.

Nous avons aussi déterminé une borne inférieure pour le nombre d'unités mobiles susceptible d'être choisies pour la réplication. Pour avoir la garantie d'atteindre les objectifs de minimisation de délai et d'augmentation du débit efficace, nous avons modélisé l'ensemble du réseau et ses entités comme un système asservi au nombre d'unités mobiles optimales pour la réplication partielle de données. Ainsi, un mécanisme d'expansion est mis en place pour remplacer toute unité hébergeant des réplicas en cas de défaillance.

La compression de données est utilisée pour réduire le temps de réplication des données et augmenter le nombre d'objets stockés sur les unités mobiles choisies pour héberger les réplicas du serveur de données.

L'ensemble du modèle que nous avons dénommé *QRScheme* a fait l'objet d'une comparaison avec le modèle de mise en antémémoire systématique dans lequel toutes les

unités mobiles disposent d'un espace antémémoire, d'une configuration dans laquelle il n'existe aucun mécanisme de mise en antémémoire ou de réplication de données et enfin d'une configuration existant dans la littérature. Pour évaluer les performances du modèle de réplication partielle de données que nous avons proposé, nous avons implémenté et simulé le modèle, ainsi que les configurations de comparaison au moyen d'un simulateur pour réseau sans fil.

De prime abord, nous avons implémenté la configuration sans aucun mécanisme de mise en antémémoire ou de réplication de données. Par la suite, nous avons implémenté le modèle de mise en antémémoire et celui existant dans la littérature, et enfin, le modèle de réplication partielle de données.

Des indices de performance ont été définis ainsi que des facteurs qui influencent le modèle. Les variations de ces facteurs à travers des niveaux préfixés ont fourni un ensemble de résultats montrant la performance du modèle de réplication partielle de données, comparativement aux autres configurations étudiées. Dans la majeure partie des cas étudiés, le délai moyen d'accès aux objets est nettement réduit, et le débit efficace amélioré. De plus, la charge du serveur est considérablement réduite et distribuée sur l'ensemble des unités mobiles hébergeant des réplicas de données du serveur. La mise en antémémoire n'est donc plus forcément la meilleure stratégie de partage de données dans les réseaux ad hoc. La réplication de données jusque-là quasi inexplorée dans les réseaux ad hoc est donc une stratégie dont l'usage est à approfondir dans ces types de réseaux.

5.2 Limitations des travaux

En dépit des résultats intéressants obtenus avec la stratégie de réplication partielle de données proposée dans ce mémoire, il persiste certaines limitations sur lesquelles nous mettons l'emphase. En effet, les résultats obtenus qui assurent la performance de la réplication partielle de données par rapport aux autres schémas sont générés avec l'usage d'un protocole de routage réactif, nommément AODV. Les mêmes expériences effectuées avec un protocole proactif tel qu'OLSR produisent une

dégradation de performance de la stratégie de réplication proposée en ce qui concerne le délai moyen d'accès aux objets. Donc, le problème du routage constitue toujours un goulot d'étranglement dans les réseaux ad hoc. Le modèle de réplication partielle proposée, n'est donc pas indépendant du protocole de routage utilisé. Il faudra donc, pour le rendre indépendant du protocole de routage, effectuer des améliorations en tenant compte des réalités du routage dans les réseaux ad hoc. D'autre part, lorsque les nœuds du réseau deviennent très mobiles, on note également une dégradation du débit efficace au-delà d'une certaine valeur (typiquement 10 m/s) de la vitesse de ces nœuds.

5.3 Travaux futurs

Nous avons proposé une stratégie de réplication partielle de données combinée avec la mise en antémémoire classique. Cependant, il reste à appliquer les mécanismes d'invalidation d'antémémoires et à réaliser la mise en place de mécanismes de mise à jour des données répliquées sur les unités mobiles choisies.

Il importe aussi de rendre le mécanisme de gestion des pannes de QR plus rapide, pour réduire le temps de dégradation des indices de performance.

Une autre indication de recherche future, serait d'étudier et de concevoir un vivier des mécanismes de remplacement d'objets stockés en antémémoire pour augmenter davantage la probabilité d'accès aux objets requis par les UM. On pourrait envisager par exemple de combiner plusieurs de ces mécanismes de remplacement d'objets suivant le mode de fonctionnement de l'UM (libre, déconnecté, etc.). De manière spécifique, un mécanisme de remplacement efficace combiné avec la réplication partielle garantirait des performances encore meilleures.

Enfin, il serait aussi intéressant de réaliser un déploiement réel à l'aide d'un réseau ad hoc connecté à un point d'accès aux données, pour assurer une validation complète du modèle.

BIBLIOGRAPHIE

[1] HASSANEIN, H., ZHENGANG, L., MARTIN P., "Performance Comparison of Alternative Web Caching Techniques" Proceedings of the 7th IEEE International Symposium on Computers and Communications, pp. 213 - 218, 1-4 July 2002.

[2] TEWARI, R., DAHLIN, M., VIN, H.M., KAY, J.S., "Design considerations for distributed caching on the Internet," Proceedings of 19th IEEE International Conference on Distributed Computing Systems, pp. 273 – 284, 1999.

[3] WOLMAN, A., VOELKER, M., SHARMA, N., CARDWELL, N., KARLIN, A., LEVY, H.M., "On the scale and performance of cooperative Web proxy caching" Proceedings of the seventeenth ACM symposium on Operating systems principles, Charleston, South Carolina, United States , pp. 16 – 31, 1999.

[4] PIERRE, S., "Réseaux et systèmes informatiques mobiles Fondements, architectures et applications" ISBN: 2-553-01038-9, 2ème trimestre 2003.

[5] FORMAN, G.H., ZAHORJAN, J., "The challenges of mobile computing" IEEE Journal of Computer Sciences, Vol. 27, Issue 4, pp. 38 - 47, April 1994.

[6] BARBARA, D., IMIELINSKI, T., "Sleepers and workholics: Caching Strategies in mobile environments," Proceedings of ACM SIGMOD'94, pp. 1-12, 1994.

[7] YEUNG, M.K.H., KWOK, Y-K., "Wireless cache invalidation schemes with link adaptation and downlink traffic" IEEE Transactions on Mobile Computing, Vol. 4, Issue 1, pp. 68 – 83, Jan-Feb 2005.

[8] BRESLAU, L., CAO, P., FAN, L., PHILLIPS, G., SHENKER, S., " Web Caching and Zipf-like Distributions: Evidence and Implications," Proceedings of Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies., New York, USA, Vol. 1, pp.126-134, March 21-25 1999.

[9] TAN, K-L., CAI, J., OOI, B. C., "An Evaluation of Cache Invalidation Strategies in Wireless Environments", IEEE Transactions on Parallel and Distributed Systems, Vol. 12, Issue 8, pp 789-806, August 2001.

[10] NUGGEHALLI, P., SRINIVASAN, V., CHIASSERINI, C., RAO, R., "Energy-Efficient Caching Strategies in Ad Hoc Wireless Networks," Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing, Annapolis, Maryland, USA, pp. 25 – 34, 2003.

[11] CAO, G., "Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," IEEE Transactions on Knowledge and Data Engineering, Vol. 51, pp. 1251-1265, 2003.

[12] CAO, G., YIN, L., DAS, C.R., "Cooperative Cache-based data access in ad hoc networks," IEEE INFOCOM, Vol. 37, Issue 2, pp. 32 - 39, Feb 2004.

[13] BARBARA, D., "Mobile Computing and Databases: A survey," IEEE Transactions on Knowledge and Data Engineering, Vol. 11, Issue 1, pp.108 – 117, Jan.-Feb. 1999.

[14] HARA, T., "Cooperative Caching by Mobile Clients in Push-based Information Systems," Proceedings of the 11th ACM International Conference on Information and Knowledge Management, Virginia, USA, pp. 186 – 193, November 2002.

[15] HARA, T., "Replica Allocation Methods in Ad hoc Networks with data update," ACM Mobile Networks and Applications, Vol. 8, Issue 4, pp. 343 – 354, 2003.

[16] FRANKLIN, M.J., CAREY, M.J., LIVNY, M., "Transactionnal Client-Server Cache Consistency: Alternatives and Performances," ACM Transactions on Database Systems, Vol. 22, Issue 3, pp.315 – 363, September 1997.

[17] YIN, L., CAO, G., "Balancing the tradeoffs between Accessibility and Query Delay in Ad Hoc Networks," Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on 18-20 Oct. 2004 pp 289 – 298

[18] CHAN, B.Y., SI, A., LEONG, H.V., "Cache management for mobile databases:

design and evaluation," Proceedings of 14th IEEE International Conference on Data Engineering, pp. 54-63, Feb. 1998.

[19] CAO, G., "On improving the performance of Cache Invalidation in Mobile Environments", ACM/Kluwer Mobile Networks and Application (MONET), Vol. 7, Issue. 4, pp. 291-303, August 2002.

[20] JING, J., ELGARAMID, A., HELAL, A., ALONSO, R., "Bit Sequences: An Adaptive cache Invalidation Method in Mobile Client/Server Environments," ACM-Baltzer Mobile Networks and Applications, Vol. 2, Issue. 2, pp.115 - 127, 1997.

[21] KOSSMANN, D., "The state of the art in distributed query processing" ACM Computing Surveys (CSUR), Vol.32 , Issue 4, pp. 422 – 469, December 2000

[22] PENG, W-C., CHEN, M-S., "Design and performance studies of an adaptive cache retrieval scheme in a mobile computing environment," IEEE Transactions on Mobile Computing, Vol. 4, Issue 1, pp. 29 – 40, Jan.-Feb. 2005.

[23] WAP-175-CacheOp-20010731-a, version 31-Jul-2001 http://www.wapforum.org/

[24] YU, J.Y., CHONG, P.H.J., "A survey of clustering schemes for mobile ad hoc networks" IEEE Communications Surveys & Tutorials, Vol.7, Issue 1, pp. 32 – 48, 2005.

[25] NAGASHIMA, K., TAKATA, M., WATANABE, T., "Evaluations of a directional MAC protocol for ad hoc networks" Proceedings. 24th International Conference on Distributed Computing Systems Workshops, pp. 678 – 683, 2004.

[26] KINSNER, W., GREENFIELD, R.H., "The Lempel-Ziv-Welch (LZW) data compression algorithm for packet radio" WESCANEX '91 IEEE Western Canada Conference on Computer, Power and Communications Systems in a Rural Environment pp.225 – 229, 29-30 May 1991.

[27] Qualnet Simulator http://www.scalable-networks.com/help/index.html

[28] YIN, L., CAO, G., CAI, Y., "Target-Driven Cache Replacement for Mobile Environments," IEEE Journal of Parallel and Distributed Computing, Vol. 65, pp. 583-594, 2005.

[29] LIM, S., LEE, W., CAO, G., DAS, C., "Performance Comparison of Cache Invalidation Strategies for Internet-based Mobile Ad Hoc Networks," IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS), pp. 104 – 113, 25-27 Oct. 2004.

[30] YIN, L., CAO, G., "Supporting Cooperative Caching in Ad Hoc Networks," Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 4, pp. 2537 - 2547, March 2004.

[31] HARA, T., "Effective Replica Allocation in Ad Hoc Networks for Improving Data Accessibility," Proceedings of IEEE Twentieth Annual Joint Conference of the Computer and Communications Societies, Vol. 3, pp. 1568 – 1576, 22-26 April 2001.

[32] ARTAIL, H., SAFA, H., PIERRE, S., "Database Caching in MANET Based on Separation of Queries and Responses," Proceedings of IEEE WiMob'2005, Vol. 3, pp. 237-244, September 2005.

[33] BERNERS-LEE, T., MIELSEN, H.F., "Propagation, Caching and replication on web," http://www.w3.org/Propagation

[34] CHEN, W., MARTIN, P., HASSANEIN, H.S., "Caching dynamic content on the Web," IEEE Canadian Conference on Electrical and Computer Engineering, Vol. 2, pp. 947–950, 2003.

[35] HU, Q., LEE, H., "Adaptive cache Invalidation Methods in Mobile Environments," Proceedings of the Sixth IEEE International Symposium on High Performance Distributed Computing, pp. 264-273, Aug.1997.

[36] MOHAN, C., "Caching Technologies for Web Applications," Proceedings of the 27th International Conference on Very Large Databases, p.726, September 11 - 14, 2001.